SOFTWARE&RIVISTE D.O.S.

Amiga Transactor

PER

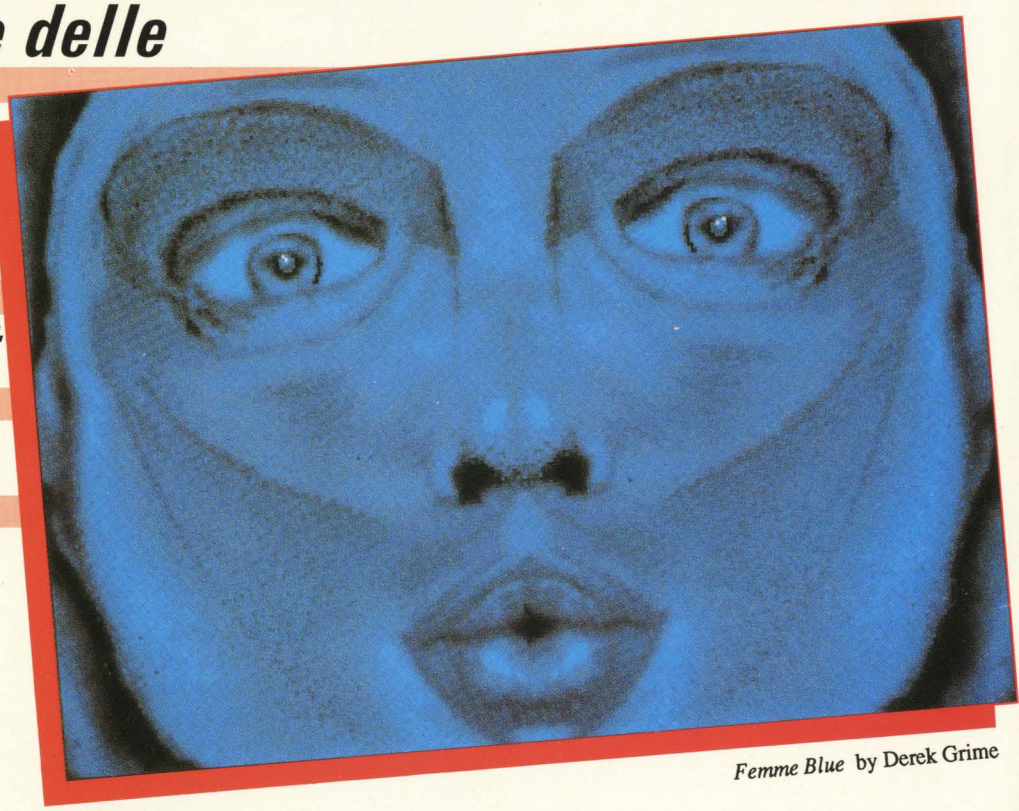
EDIZIONE ITALIANA

L. 7.000 - S.FR. 10,50

MARZO-APRILE 1989: NUMERO 3

LA RIVISTA PER I PROGRAMMATORI DI AMIGA

- 2ª Conferenza Europea degli sviluppatori
- I cambiamenti nei Font
- Come richiamare il CLI dall'AmigaBasic
- Programmazione della Shell
- Come costruire una scheda con 68020 per Amiga
- Programmare in Modula-2 con Amiga
- Scopriamo i messaggi del Guru
- Come creare delle Run-Time Library
- Breackpoint parte terza



Femme Blue by Derek Grime

GRUPPO EDITORIALE JACKSON
AREA CONSUMER

TAXE PERCUE (TASSA RISCOSSA) MILANO FERR. CORR. — GRUPPO IV/70

GRUPPO EDITORIALE JACKSON

IN COLLABORAZIONE CON

AMSTRAD

PRESENTA

LA GRANDE ENCICLOPEDIA DI INFORMATICA PER RAGAZZI

- IMPARIAMO A PROGRAMMARE
- COME È FATTO E COME FUNZIONA
- GIOCHI, GIOCHI, GIOCHI
- TECNOLOGIA E APPLICAZIONI
- SI FA... NON SI FA

L'informatica ti appassiona? vorresti approfondirla per studiare, giocare, curiosare?
Inizialmente, per potersi orientare, è preferibile una guida giovane ma rigorosa e completa, senza risultare difficile o noiosa, da leggere tutta d'un fiato, corredata da centinaia di illustrazioni a colori.

VINCI 30 FAVOLOSI COMPUTER AMSTRAD CPC 6128

Questa è LA GRANDE ENCICLOPEDIA DI INFORMATICA PER RAGAZZI. Potrai entrare nei segreti del personal computer: storia, termini, concetti fondamentali, hardware, funzionamento e tanti programmi software in linguaggio Basic. JACKSON, in collaborazione con AMSTRAD, ti svela i segreti dell'informatica in fascicoli settimanali, **nella tua edicola**, a sole lire 2.500, da rilegare in 2 splendidi volumi illustrati.
Ma non è tutto!
Assieme al grande poster del Basic a colori, sul primo fascicolo scoprirai come vincere uno splendido computer AMSTRAD CPC 6128.

Sganciati dall'ordinario, entra nello straordinario!

Il terzo mondo

Le lettere arrivate in redazione sono molto chiare.
La situazione italiana dell'informatica di consumo è veramente disastrosa. Siamo un paese, da questo punto di vista, appartenente al terzo mondo. Non mi riferisco assolutamente al livello degli utenti e dei programmatori ma alle strutture esistenti e al modo di pensare di chi vi gravita attorno. Stentate a crederci?
Cominciamo dalla lingua.
E' inutile nascondere che l'ostacolo maggiore, per chi voglia approfondire le proprie conoscenze, sta proprio nella mancanza o nella difficile reperibilità di materiale in italiano. Quanti hanno acquistato un Amiga in Italia e hanno ricevuto dei manuali in Italiano? Noi per primi sappiamo quali siano le difficoltà nel tradurre del materiale tecnico, ma molti lettori si chiedono se il ritardo o la mancanza non sia dovuta a cattiva volontà. Comunque i manuali adesso sono in italiano (beati gli ultimi...). Resta il fatto che tutto il resto rimane in inglese, a partire dai manuali tecnici per finire ai libri e alle istruzioni dei giochi importati. Le istruzioni dei programmi non vengono mai, o quasi mai, tradotte per il fatto che gli introiti provenienti dalle vendite non sarebbero in grado di coprire le spese aggiuntive.
Perché le vendite sono poche? Per i pirati? Per mancanza di operatori preparati? Per mancanza di iniziative adeguate? Per il momento non ci interessa analizzare le cause ma evidenziare i problemi.
Andiamo avanti, quindi.
L'assistenza post-vendita.
Chi non ha avuto problemi dopo che ha acquistato un computer, l'ha portato a casa e l'ha usato alzi la mano. Purtroppo non siamo in grado come Celentano di far spegnere il video a milioni di persone per confermare la veridicità delle proprie affermazioni, probabilmente non si va lontano dal vero affermando che per le ditte produttrici, i distributori e i venditori, un utente che ha effettuato un acquisto è un utente morto, in quanto non comprerà più la stessa cosa. In effetti l'unica cosa di cui un utente ha bisogno dopo che ha comprato sono solo informazioni, consulenza, riparazioni e software per far funzionare l'amato silicio. Il negoziante, spesso, non è in grado di dare nessuna delle prima tre, mentre per l'ultima c'è "l'ultimo copiatore che copia tutto". "In fondo che male c'è, non ti fornisco informazioni, consulenza e riparazioni ma ti regalo dieci programmi che da soli valgono mezzo milione. Per le riparazioni ci sono i centri di assistenza autorizzati, per le informazioni ci sono fior di libri - Ho il ROM Kernel fotocopiato, lo vuoi? - e per la consulenza leggi le riviste."
Bene, ma i centri di assistenza funzionano? I libri esistono? Le riviste sono serie?
Ora, mi sembra di sentire un coro come quello dei politici che siedono il venerdì sulla poltrona bollente di Diogene (una trasmissione del TG2 in onda nel TG dell'una) dicendo che hanno già pensato alla soluzione, che è pronto un disegno di legge, che i soldi sono già stati stanziati, ecc.
Ma proviamo a girare la frittata. Tutte le attività commerciali collegate con il mondo dei computer domestici seguono la legge della domanda e dell'offerta. Se nessuno comprasse questa rivista probabilmente entro pochi numeri chiuderebbe, così come se ci fossero 10000 richieste di manuali ROM Kernel, qualcuno si affretterebbe a tradurli. Se nessuno comprasse software piratato, o meglio, se tutti comprassero software originale riceverebbero istruzioni in italiano. Se si esigesse un po' di assistenza post-vendita, a costo di cambiare negoziante e pagare qualcosa in più, probabilmente si potrebbero dormire sonni tranquilli. Se...
Ma allora, di chi è la colpa?

Marco Ottolini

DIRETTORE RESPONSABILE
Paolo Reina

DIRETTORE EDITORIALE
Daniele Comboni

DIRETTORE TECNICO
Marco Ottolini

TRADUZIONI E REDAZIONE
Franco Bellotti, Giuseppe Bravo, Leonardo Fei,
Paolo Toccaceli, Giovanna Traversa

GRAFICA
Cristina Turra

IMPAGINAZIONE ELETTRONICA
Roberto Gibertini

STAMPA
Grafica F.B.M. Gorgonzola (MI)

AUTORIZZAZIONE ALLA PUBBLICAZIONE
Trib. di Milano n. 866 del 20/12/88

AREA CONSUMER
PUBLISHER
Filippo Canavese

DIREZIONE, REDAZIONE, PUBLICITA' E AMMINISTRAZIONE
via Rossellini, 12 - 20124 Milano
Tel. 02/66800000-66800161-66800272-66800238
Telex: 333436 GEJIT
Telefax: 02/6948238

CONCESSIONARIO ESCLUSIVO
SODIP - via Zuretti, 25 - 20125 Milano

ABBONAMENTI E MAGAZZINO
via Gasparotto, 15 - 20092 Cinisello B. (MI)
Tel. 02/61222527-6187736

SEDE LEGALE
via Pietro Mascagni, 14 - 20122 Milano

PREZZO DELLA RIVISTA: L. 7000
NUMERI ARRETRATI: L. 14000
ABBONAMENTO ANNUO (6 numeri): L. 34000
ESTERO: L. 68000

PUBLISHER
Richard Evers

EDITORS
Nick Sullivan, Chris Zamara

ASSISTANT EDITOR
Malcolm O'Brien

EDITORIAL ASSISTANT
Moya Drummond

CUSTOMER SERVICE
Renanne Turner

CONTRIBUTING WRITERS
S. Ahlstrom, S. Ballantine, C.B. Blish, J. Butterfield, Betty Clay,
D. Curtis, M. Dillon, A. Finkel, C. Innes, C. Gray, P. Kivolowitz,
R. Mariani, B. Nesbitt, R. Peck, L. Phillips, B. Rakosky,
J. Toebes, V.A. Wagner, D. Wood

Transactor UK Ltd
David H. Beatty
(UK Publisher)

per *AMIGA*
Transactor
La rivista dei programmatori di Amiga

Sommario

Associato al

Consorzio Stampa Specializzata Tecnica

Testata aderente al C.S.S.T. non soggetta a certificazione obbligatoria in quanto la presenza pubblicitaria è inferiore al 10%

Lettere...

Consigli e critiche 1

Benissimo!
Volete che i lettori intervengano alla vostra rubrica "LETTERE..."?
Non volete i complimenti?
Eccovi accontentati! come potevate pretendere che qualcuno mandasse una lettera a una rivista della quale non era certa l'uscita del secondo numero?!?
Va bene la periodicità bimestrale, ma che al 30 gennaio nessun edicolante avesse idea di cosa si stesse parlando... (testuale: "Ci hanno ritirato gli invenduti il 23, ma forse il secondo numero non ci sarà!").
Per favore, curate meglio la distribuzione!!
La vostra rivista si pone a un livello estremamente competitivo rispetto alle altre, soprattutto per i contenuti e gli obiettivi, ma...

1) L'editoriale del n.2 (finalmente reperito oggi 12-2-89!) si riferisce a un certo MORO. Siamo in tanti nella sua condizione: troppi. Ma vi siete chiesti quali possono essere le difficoltà incontrate dagli aspiranti programmatori? Io non sono stupido (almeno lo spero!), lavoro come agente di cambio per un noto istituto bancario e sono nello stesso tempo il responsabile per i problemi di carattere "informatico" che si riscontrano in Sala Cambi: costituisco in pratica l'interfaccia tra gli agenti e i programmatori veri e propri. E' il mio mestiere individuare un problema, analizzarlo e predisporre le basi per la sua soluzione. Frustrazione: con l'Amiga più di qualche "programmino" e tanti bei giochi non riesco a fare! Ma è proprio solo colpa mia o di come è stata impostata la documentazione su questa macchina? I manualetti forniti in dotazione sono estremamente superficiali, e quelli che forse sarebbero utili sono difficilmente reperibili (in inglese, forse in italiano non esistono neanche) e comunque piuttosto costosi. Il Basic sembra molto potente ma a parte le incoerenze mostruose che si possono incontrare nel suo uso (per esempio, provate a usare l'istruzione CLEAR 280000, stoppare il programma e rilanciarlo: OUT OF MEMORY! Con 1Mb che cosa vuol dire?!?) non mi sembra il più adatto per utilizzare al meglio tutte le potenzialità di Amiga.
Il C!! Ecco la chiave di volta! ... e quale libro mi consigliate voi? La "Bibbia" di Kernighan e Ritchie o "Programmare in C" di Hunt (in cui sono frequentissimi i riferimenti al PL1, ADA, COBOL... e se uno non li conosce?)?
Voi non vi metterete certo a fare un corso di C (molto male!) però esaminate le ARP.Library, così se uno non ne sapeva niente prima, dopo... sarà lo stesso!!!
Ora, non credo di sbagliare se affermo che c'è veramente bisogno di un "corso di C": quello che serve però non è un corso stile "La mia Chitarra" (senza offesa) ma di qualcosa di interattivo, una sorta di dialogo tra voi e noi, partendo dall'inizio (tipi di variabili, l-valori, puntatori) fino alle operazioni più complesse (gestione file su disco, gestione della stampante) con riferimenti precisi ma soprattutto chiari a queste "fantomatiche" funzioni di libreria. Conosco più di una ventina di possessori di Amiga, tutti più o meno con le mie stesse idee, chi interessato alla grafica, chi al sonoro ma tutti convinti che l'Amiga sia più di una console per videogame.

2) Sul n.1 è apparso un bellissimo articolo in cui si affermava che "programmare l'Amiga è facile". Come prefazione mi è sembrata eccellente ma sul n.2 non vedo traccia del I capitolo.

3) Spiegatevi sempre in modo chiaro come se davanti aveste solo dei bambini, ansiosi di apprendere ma pur sempre bambini.

Caro Marco O. il coraggio non ci manca, ma non sappiamo da dove cominciare.
Possiamo anche riuscire a collaborare con voi, ma suggeriteci da dove partire per esplorare questo benedetto Amiga.
Infine mi scuso, se ho dato l'impressione di un'eccessiva aggressività: dico quel che penso anche se poi devo sostenerne le conseguenze, o veder buttate le parole al vento (voi potreste anche non pubblicare questa letterina, no?!)
Ora comunque sono io che lo dico a voi, da come è steso l'editoriale del N.2 si capisce che sapete la condizione in cui ci troviamo: coraggio!
NON LASCIATECI ANNEGARE NEI VIDEOGAME!!!

Enrico Ferrante

Consigli e critiche 2

Sono un lettore molto soddisfatto del vostro lavoro, che mi permette di conoscere meglio il computer che uso 6 ore al giorno, 7 giorni alla settimana, e che ritengo sotto sfruttato in alcuni campi applicativi come per esempio le tecniche di gestione dei progetti, la connessione in rete di lavoro, e la gestione di basi dati tipo HyperText (anche se ho visto SoftwoodFileIIsg e mi sembra che faccia parecchio da sé). Con questa lettera intendo chiarirvi solo poche cose passate per la testa usando il mio Amiga, ma non stupitevi se salto di palo in frasca, non ho molto tempo per scrivervi, sapete sotto esame...
"Programmare Amiga è facile" (v. Editoriale n.2), sarà anche vero visto la mia scarsa esperienza, quello che certo conoscete è quanto siano realmente utili i manualetti forniti con la macchina al momento dell'acquisto per un programmatore ("... ci sono le librerie per sfruttare tutta la potenza di Amiga, ..." ma cosa c'è nelle librerie?), e quanto siano diffuse le espansioni e i pezzi di ricambio.
Il 7 marzo ho incontrato per la terza volta in un Commodore Point della mia città lo stesso signore che ha comprato a Padova un modello di Amiga con la tastiera FRANCESE. Nulla di male, solo che perfino cambiare tastiera è difficile, sembra che in tutti i riparatori autorizzati Commodore in Italia non ci sia una sola tastiera di ricambio, né i tasti da sostituire; in compen-

Fondamentali per lo studio, il lavoro e l'aggiornamento

i dizionari enciclopedici di:

Matematica
Fisica • Chimica
Informatica • Meccanica
Astronomia • Biologia • Geologia
Ragioneria Generale
Ragioneria Applicata • Elettronica

IN EDICOLA
OGNI MESE QUATTRO ARGOMENTI
A LIRE 14.000 CIASCUNO

Conoscenza e informazione, chiarezza e rigore scientifico in 15.000 termini e oltre 650 illustrazioni, tabelle e schemi.

Fondamentali per il nostro tempo.

GRUPPO EDITORIALE JACKSON

I Servizi
di

Amiga Transactor offre una serie di servizi per agevolare i propri lettori nel reperimento di software e materiale utile alla programmazione.
È disponibile l'intera libreria di dischetti di pubblico dominio curata da Fred Fish. Ogni dischetto contiene numerosi programmi e utility, spesso corredati da listati sorgenti e commenti degli autori.
Per districarsi fra le centinaia di programmi disponibili nei dischi di Fred Fish, è stato creato un apposito catalogo di 20 pagine. Tale elenco riporta, divisi per categoria e in ordine alfabetico, tutti i programmi presenti, completandoli con informazioni quali la descrizione della funzione, l'autore, il numero di versione, la disponibilità del sorgente e il disco nel quale sono contenuti. I dischetti possono essere ordinati contrassegnando i numeri desiderati, purché la quantità sia un multiplo di cinque. Per ordini amministrativi saremo costretti a non accettare ordini che non soddisfino questa regola. Tutto il materiale, a esclusione dei listati pubblicati sulla rivista, viene fornito nella versione originale americana, senza traduzione o modifiche.
A ogni numero della rivista corrisponde un disco chiamato "AmiTrans Disk" che contiene tutti i listati pubblicati su quel numero, nonché i corrispondenti eseguibili e tutti gli altri programmi di pubblico dominio menzionati negli articoli.

BUONO D'ORDINE

Completa il buono d'ordine (o una sua fotocopia) e spedire in busta chiusa a: I servizi di Transactor per Amiga - Via Rosellini, 12 - 20124 Milano

Si può allegare: assegno, contanti o fotocopia della ricevuta di versamento c/c n. 11666203 intestato a Gruppo Editoriale Jackson.
Non si effettuano spedizioni contrassegno

Desidero ricevere i seguenti articoli; contrassegnare con una X i numeri di Fish disk desiderati (a gruppi di 5)

Nota: Il disco 164 non è disponibile

☐ 1	☐ 12	☐ 23	☐ 34	☐ 45	☐ 56	☐ 67	☐ 78	☐ 89	☐ 100	☐ 111	☐ 122	☐ 133	☐ 144	☐ 155	☐ 167
☐ 2	☐ 13	☐ 24	☐ 35	☐ 46	☐ 57	☐ 68	☐ 79	☐ 90	☐ 101	☐ 112	☐ 123	☐ 134	☐ 145	☐ 156	☐ 168
☐ 3	☐ 14	☐ 25	☐ 36	☐ 47	☐ 58	☐ 69	☐ 80	☐ 91	☐ 102	☐ 113	☐ 124	☐ 135	☐ 146	☐ 157	☐ 169
☐ 4	☐ 15	☐ 26	☐ 37	☐ 48	☐ 59	☐ 70	☐ 81	☐ 92	☐ 103	☐ 114	☐ 125	☐ 136	☐ 147	☐ 158	☐ 170
☐ 5	☐ 16	☐ 27	☐ 38	☐ 49	☐ 60	☐ 71	☐ 82	☐ 93	☐ 104	☐ 115	☐ 126	☐ 137	☐ 148	☐ 159	☐ 171
☐ 6	☐ 17	☐ 28	☐ 39	☐ 50	☐ 61	☐ 72	☐ 83	☐ 94	☐ 105	☐ 116	☐ 127	☐ 138	☐ 149	☐ 160	☐ 172
☐ 7	☐ 18	☐ 29	☐ 40	☐ 51	☐ 62	☐ 73	☐ 84	☐ 95	☐ 106	☐ 117	☐ 128	☐ 139	☐ 150	☐ 161	
☐ 8	☐ 19	☐ 30	☐ 41	☐ 52	☐ 63	☐ 74	☐ 85	☐ 96	☐ 107	☐ 118	☐ 129	☐ 140	☐ 151	☐ 162	
☐ 9	☐ 20	☐ 31	☐ 42	☐ 53	☐ 64	☐ 75	☐ 86	☐ 97	☐ 108	☐ 119	☐ 130	☐ 141	☐ 152	☐ 163	
☐ 10	☐ 21	☐ 32	☐ 43	☐ 54	☐ 65	☐ 76	☐ 87	☐ 98	☐ 109	☐ 120	☐ 131	☐ 142	☐ 153	☐ 165	
☐ 11	☐ 22	☐ 33	☐ 44	☐ 55	☐ 66	☐ 77	☐ 88	☐ 99	☐ 110	☐ 121	☐ 132	☐ 143	☐ 154	☐ 166	

DESCRIZIONE

☐ Fish Disk	Lit. 35.000 per gruppo di cinque
☐ Catalogo	Lit. 15.000 aggiornato fino al Fish 138
☐ AmiTrans Disk #1	Lit 15.000
☐ #2	Lit 15.000

Tutti i prezzi sono da intendersi IVA inclusa e spese di spedizione comprese.

Cognome .....

Nome .....

Via .....

Cap. ..... Città .....

Prov. ..... Tel. .....

Firma .....

(se minorenne quella di un genitore)
Gli ordini non firmati non verranno evasi.

so i pezzi di ricambio presenti sul mercato seguono la politica del 64: al costo di otto-dieci ricambi nastro-stampante, mi posso permettere una nuova stampante; comprare espansione+2 drive originali Commodore costa come un Amiga 500 "nudo", e si può continuare ancora,... E' solo questioni di scelte personali, ma del negoziante! Lui/lei ti propone sempre quello che ha "in casa".

Invece, per programmare Amiga come si merita, mi sono comprato il mitico (originale e non foto-rubato!) Amiga Intuition Reference Manual (costo listino £54000), nel quale sono spiegati alcuni aspetti del sistema operativo. Nondimeno è un libro che ha il suo peso (870 gr circa) nella cultura necessaria per fare dei programmi più Amig-hevoli anche se trovo che molto viene divulgato in pillole nelle riviste come la vostra. Insomma l'ho trovato interessante ma caruccio. Penso che dal 1987 (data della stampa del manualone, basato sulla versione 1.1) ci siano state modifiche al sistema operativo (è uscita già la versione 1.3), ma non c'è neppure una cartolina di sconto sul nuovo manuale con lo stesso titolo, seconda edizione, per quando verrà stampata... Pensate che pretenda troppo, o che sia naturale attendersi una qualche forma di agevolazione per avere gli aggiornamenti?

A proposito di aggiornamenti...

La nostra cara ditta fornitrice di Amiga, insieme con il Workbench, l'AmigaBasic Microsoft, il MicroEmacs e altro, non ci potrebbe mettere un disco di supporto ai programmatori di pubblico dominio, con il data base degli identificatori usati nelle LIBRARY per i sorgenti dei programmi e loro definizioni, parametri, esempi? Per usarlo non sarebbe necessario un DB IV, magari solo una versione estesa di More. Questo potrebbe essere diffuso molto di più e molto più rapidamente tra gli utenti programmatori (onesti e pirateschi) dell'enciclopedia dei manuali tecnici, con vantaggi sullo sviluppo di programmi importanti per la penetrazione del mercato; si potrebbero tradurre in italiano abbastanza velocemente, ne sapete qualcosa al proposito. La proposta vale anche per voi, CBM Copyright permettendo.

I vostri dischi, e quelli di pubblico dominio (FISH per esempio), non potrebbero venire distribuiti anche attraverso i Jackson Center e/o i Commodore Point? Comprare un programma commerciale copiato a 15.000 lire è piratato, un disco con alcuni o tanti programmi PD documentati alla stessa cifra è relativamente onesto, non trovate? Questo perché i programmi PD, che non sono pochi, non si trovano come i giochi piratati, insomma fate voi: io il sasso l'ho buttato.

"Buttate" una mezza pagina della vostra rivista, in caratteri piccoli-piccoli per descrivere in dettaglio i file contenuti nei dischi AmiTrans, e/o presentate almeno un paio di Fish-Disk alla volta. Titolo della rubrica: >KompattatI<, oppure facendo il verso a una che leggo su BYTE: "Some Lens Required".

A richiesta degli abbonati, per modem o per lettera (poste permettendo), fate un gruppo di discussione dei problemi di programmazione di Amiga (sul tipo di BIX in America), almeno fateci un pensierino serio, siete voi la Jackson, o no?

Massimo Rainato

Consigli e critiche 3

Cara redazione di Transactor, ho visto la vostra rivista a dicembre e, non appena ho letto che era interamente dedicata ai programmatori, l'ho subito comprata pensando che finalmente fosse venuta la buona volontà a qualcuno di occuparsi seriamente di questa macchina fantastica.

Devo dire che il primo numero non era male, ma ora che ho letto il secondo sono proprio deluso.

In ogni articolo ci sono solamente programmi in linguaggio C mentre di linguaggio Assembly non si dice nulla né tantomeno si fanno esempi. Ora, dato che in Italia il C è probabilmente il linguaggio più odiato e meno conosciuto, non capisco come mai non rivolgiate la vostra attenzione all'Assembly che è molto più conosciuto, più facile e, diciamo la verità, più potente.

Spero che non giudichiate questa lettera come dettata dell'egoismo di un medio programmatore in Assembly, ma condizionata dai motivi che vi ho prima elencati.

Dopo tutto questo discorso quindi, voglio sperare che cambierete orizzonte poiché penso che Transactor potrebbe diventare la migliore rivista sinora fatta per Amiga.

Pierfabio Campolongo

Tra tutte le lettere arrivate abbiamo scelte queste tre perché sembrano riassumere al meglio il contenuto anche delle altre. Prima di rispondere permettetemi di complimentarvi con voi per aver risposto al mio appello del secondo numero, riguardante una mancanza di corrispondenza con i lettori, e di tirarvi nello stesso tempo le orecchie per non aver mandato nemmeno UN BIT!

Sono arrivate tante critiche costruttive, altre meno, pochi complimenti (grazie) che ci hanno permesso di inquadrare un po' meglio il tipo di pubblico che ci legge. Identificando nel Moro (vedi Editoriale N.2) il lettore medio non ci eravamo molto sbagliati: tutti vogliamo sapere di più sul come funzioni Amiga e su come si possa sfruttare al meglio.

La prima cosa di cui si dovrebbe disporre è una "visione globale" di Amiga. Con il termine "visione globale" intendo una conoscenza diffusa e non approfondita, ma chiara al tempo stesso, di come sia strutturato Amiga, sia dal punto di vista hardware che software. Con il Commodore 64 (non neghiamolo, il 90% degli utenti Amiga ha un C64 nell'armadio) era molto semplice farsi una visione globale: esiste il 6510, il SID per il suono con 4 forme d'onda e i registri per l'ADSR, il VIC II...

Con Amiga farsi una visione globale è molto più difficile per diversi motivi. Primo fra tutti il team che lo progettò. Non si limitarono a creare dei chip custom (nel qual caso saremmo qui a dire: "Amiga ha il 68000, Paula, Denise, ecc.") ma ci vollero aggiungere anche un supporto software per il multitasking. Se Amiga fosse una macchina monotask sarebbe tutto più semplice, non come con il C64, ma comunque più semplice. La struttura del sistema software di Amiga è paragonabile, per certi versi, a quella di un sistema operativo come OS/2 o Unix. Non che questo ci spaventi, ma bisogna comunque considerare che le difficoltà incontrate giorno per giorno sono dovute proprio al fatto che si sta combattendo con qualcosa di più grande di noi.

Ma allora non abbiamo speranza? No, se si riesce ad avere una buona visione globale e si ha accesso a tutte le informazioni necessarie per raffinare le proprie conoscenze, si può arrivare dove si vuole (tempo permettendo). Non ci si può

aspettare però di trovare la pappa pronta, bisogna darsi da fare. I libri forse al momento sono pochi ma tra poco aumenteranno. Jackson sta pubblicando, forse sono già in libreria, una serie di libri dedicati ad Amiga, riconoscibili con una sola occhiata alla copertina. I primi sono uno dedicato alla programmazione in linguaggio assembly e uno rivolto agli amanti della grafica in 3D e alle animazioni. Entro breve dovrebbe essere disponibile un titolo sulla programmazione in C e un po' più in là nel tempo arriverà la "guida per il programmatore" di Rob Peck, l'autore del ROM Kernel.

Nel frattempo uno degli obiettivi primari di Transactor sarà fornire i lettori di una visione globale soddisfacente, già a partire dal prossimo numero. Non che non si sia già fatto, ma i contributi italiani, che finalmente ci saranno, si indirizzeranno soprattutto in questa direzione. Un'ultima cosa, non aspettatevi una serie di corsi sul linguaggio C, l'Assembly o il Basic: la visione globale è indipendente dal tipo di linguaggio che si usa per programmare.

D'accordo, e ora che ho la visione globale che cosa faccio se non so il nome delle routine contenute nelle librerie (visto che ora so che cosa sono). Il problema evidenziato da un nostro lettore in realtà non è di semplice soluzione (soprattutto nel modo da lui indicato), ma ci stiamo studiando e vedremo (forse sul N.4?) cosa salterà fuori. Non si tratterà comunque di nulla in grado di sostituire la documentazione tecnica Commodore che resta, e resterà sempre, l'unica fonte di sapere ufficiale (anche se piena di errori). La documentazione tecnica non ha comunque niente a che vedere con i manualetti forniti con la macchina. Nelle macchine vendute in questo momento si trova un manuale (in italiano!) ben strutturato ma di utilità quasi nulla per il programmatore. La scelta di Commodore di non fornire manuali più completi ci sembra però corretta: è inutile far pesare il costo di altri manuali su tutti gli utenti, meglio che li acquistino solo gli interessati. E qui veniamo a un altro punto, come fanno gli interessati ad acquistarli e a leggerli? I manuali sono in inglese e sono reperibili, non senza difficoltà, solo nelle librerie tecniche delle grandi città. Commodore non potrebbe fare qualcosa per aiutare questa fetta di utenti?

L'idea è antica quanto Amiga e ha un nome semplice: piano di supporto per gli sviluppatori. Per maggiori informazioni riferitevi all'articolo sulla conferenza degli sviluppatori di Francoforte. In attesa che qualcosa si muova non resta che seguire le strade abituali.

Tra le strade da percorre per acquisire una maggiore conoscenza non bisognerebbe trascurare il fatto di approvvigionarsi di Fish Disk. I dischi di pubblico dominio di Fred Fish sono pieni di esempi d'utilizzo delle peculiarità più nascoste di Amiga. Tra le varie directory di cui sono composti non è raro imbattersi in utility che farebbero impallidire prodotti commerciali. Bisogna anche dire che gli esempi sono sempre commentati e liberamente utilizzabili. Esisterebbero esempi a bizzeffe e quindi è forse meglio lasciare perdere. Dal prossimo numero cominceremo, seguendo alcuni consigli giunti in redazione, a pubblicare un elenco ragionato del contenuto dei Fish Disk. Stiamo ancora pianificando la cosa nei dettagli e quindi se avete ulteriori consigli fateceli pervenire al più presto. Un'ultima cosa, ricordatevi che i Fish Disk sono di pubblico dominio e quindi liberamente copiabili: spargeteli il più possibile.

Per quanto riguarda il fatto di privilegiare un linguaggio rispetto a un altro, ecco la nostra posizione. Il C deve essere considerato il vero linguaggio di programmazione ad alto livello di Amiga. Il Basic è un linguaggio importante perché tutti gli utenti lo posseggono e quindi è il più diffuso e probabilmente il più usato. L'assembly dovrebbe essere usato solo in casi particolari. Con il C64 era necessario programmare in Assembly perché con il Basic non si poteva fare niente. Con Amiga non è necessario perché già da C o da Basic si può fare quasi (attenzione, quasi) tutto. Se poi consideriamo l'efficenza il discorso cambia: deve essere scelto l'Assembly. Quindi, per quanto riguarda le traduzioni di articoli stranieri, i listati saranno nel linguaggio scelto dall'autore, per quelli di autori italiani cercheremo di proporre i listati in linguaggi diversi (C e Basic) e, in casi particolari, in Assembly. Per esempio, un articolo (ipotetico?) sulla visualizzazione di un file grafico IFF potrebbe avere due listati C e Basic completi, più una routine Assembly relativa alla sola de-compattazione. Resterebbe poi ai lettori utilizzare uno degli esempi e modificarlo a proprio piacimento. Resta anche il fatto che non trascureremo altri linguaggi come il Modula-2 (questo numero ne è un esempio) o ARexx.

I problemi avuti con la puntualità della rivista si sono ripetuti con questo numero e proseguiranno quasi sicuramente anche per la prossima. Vi assicuro che iniziare una nuova rivista con una nuova redazione non è lavoro da poco, ma le colpe non sono da attribuire tutte a noi. La nostra sorella canadese ha avuto alcuni problemi che sono culminati con il passaggio di proprietà della testata a un'altra casa editrice. Durante questo periodo, oltre a una certa incertezza sul futuro, si sono verificati ritardi nella consegna di articoli, dischetti ed altro. Ora la situazione si è normalizzata e piano piano anche noi dovremmo riuscire a rispettare le date di uscita. Quindi attendete fiduciosi l'uscita anche del prossimo numero.

...e Bit

Come rendere più piccolo un programma C Manx

Se il vostro programma non ha bisogno di leggere una lista di argomenti dal CLI o dal Workbench, potete ridurre la dimensione dell'eseguibile di oltre 700 byte impedendo che il compilatore includa (come fa di default) le funzioni di libreria create per questo scopo. Basta aggiungere nel listato le due funzioni seguenti:

```
_cli_parse() {}
_wb_parse() {}
```

Il linker vi avvertirà che queste funzioni impediscono l'uso di quelle di libreria, ma voi andate avanti senza preoccuparvi.

2ª Conferenza europea degli sviluppatori

di Marco Ottolini

Nel numero scorso di Transactor, eravamo riusciti a dare solamente la notizia del fatto che questa manifestazione si era già tenuta e che ne avremmo parlato più diffusamente sul numero successivo (questo). Ci scusiamo per non aver potuto comunicare attraverso le pagine della rivista le date della manifestazione in tempo utile perché gli eventuali interessati vi potessero partecipare, ma la notizia è arrivata a noi quasi un mese dopo l'uscita del primo numero, quindi troppo tardi visto che il secondo sarebbe uscito a conferenza già avvenuta.

La manifestazione, organizzata da Commodore per incontrare gli sviluppatori europei, si è tenuta dal 16 al 18 gennaio a Francoforte e ha visto una buona partecipazione di sviluppatori provenienti da quasi tutti i paesi europei. Ma, chi sono gli sviluppatori?

Gli sviluppatori, in teoria, dovrebbero essere tutte quelle entità (persone o società) che sviluppano software o hardware per Amiga e hanno bisogno di avere informazioni sicure su cui basare il proprio lavoro. Sempre in teoria, dovrebbero esistere tre tipologie di sviluppatori: i Commercial, i Certified e i Registred. I primi sono quelli che sviluppano per Amiga a scopo di lucro e produrranno un qualche prodotto funzionante; gli ultimi sono quelli che hanno bisogno di informazioni per divertimento o che comunque non le usano per produrre applicativi tipo Dpaint II, mentre i Certified si pongono a metà strada. Per appartenere a una di queste categorie dovrebbe essere sufficiente pagare una certa quota, variabile a seconda dell'area cui si appartiene, e dimostrare che si ha bisogno di far parte di questa elite. Le differenze tra le diverse categorie sono rappresentate da una serie di diritti/doveri cui si può/deve usufruire/sottostare. Per esempio gli sviluppatori Commercial ricevono in anteprima le versioni beta del sistema operativo così che possano modificare i prodotti già presentati oppure crearne di nuovi o più potenti per il momento in cui la nuova versione del S.O. sarà effettivamente pronta e funzionante. Tutti gli sviluppatori, poi, entrano a far parte del programma di supporto agli sviluppatori che Commodore ha predisposto per poter aiutare tutti gli sviluppatori nel loro lavoro; un esempio di questo programma è proprio l'organizzazione della conferenza di Francoforte.

Fin qui tutto a posto, ma ora come si fà a diventare sviluppatori e a raggiungere finalmente le agognate informazioni? Per il momento si aspetta, visto che in Italia (nel momento in cui scrivo) non esiste alcun programma di supporto agli sviluppatori. Il motivo di ciò ha radici antiche.

Nel lontano inverno 85-86 cominciò l'opera di reclutamento di sviluppatori italiani per Amiga da parte di Commodore. Gli sviluppatori individuati dovettero acquistare un Amiga 1000 NTSC con tutta la documentazione tecnica necessaria per un prezzo onesto, contando la quantità di materiale, il SUPPORTO promesso e il fatto che si trattava dei primi Amiga. Commodore, in quel periodo, navigava in cattive acque, tanto che alcuni osservatori predissero la scomparsa, o perlomeno il ridimensionamento, della mamma del C64. Ebbene, per un motivo o per l'altro il SUPPORTO agli sviluppatori venne a mancare e crollò così tutto. A dir la verità per un certo periodo Rick Glover, ora Support Manager di Commodore per l'Europa e sviluppatore (nonostante il nome) italiano della prima ora, cercò di ridare vita all'operazione ma poi venne promosso e andò a lavorare per Commodore in Germania. Di quel manipolo di programmatori e progettisti è rimasto ben poco: la maggior parte si è dedicato alla produzione di video e giochi per la televisione, qualcuno divulga la sua conoscenza di Amiga e qualcun'altro si dedica ad altri computer.

In mancanza di un piano organico di supporto per gli sviluppatori si è andati avanti alla giornata, distribuendo informazioni in base a criteri discutibili e in modo disordinato. Qual'è ora la situazione? Attorno a Commodore si è raccolto un certo numero di "interessati" che dovrebbero, in breve tempo, trasformarsi in sviluppatori veri e propri. L'occasione per la partenza del piano di supporto agli sviluppatori sarà data da una riunione che si dovrebbe tenere quanto prima tra la Commodore italiana e gli "interessati". Appena avremo notizie sicure sull'inizio del piano di supporto agli sviluppatori e sulle condizioni di partecipazione ne daremo notizia su queste pagine, per dar modo a tutti gli interessati di prendervi parte fin dall'inizio.

Francoforte

Dopo questa ampia parentesi, necessaria per farsi un'idea della situazione, è venuto il momento di vedere che cosa è successo a Francoforte.

La prima impressione ricavata a caldo sulla manifestazione è abbastanza difficile da interpretare: è sembrato di trovarsi di fronte a una società (Commodore) diversa, almeno nelle intenzioni. In tutti gli incontri che si sono avuti riguardanti il futuro della linea Amiga, i diversi responsabili si sono sempre affrettati a precisare che ci sono nuovi obiettivi e che si cercherà di introdurre Amiga (il 2000) in nuovi mercati. Un primo esempio del nuovo corso si è già potuto osservare anche da noi: l'aumento del prezzo di listino dell'Amiga 2000. Evidentemente si è pensato che un aumento di prezzo avrebbe fatto uscire la macchina dal regno consumer per entrare nel mercato professionale. In fondo la domanda che più spesso si sentono rivolge-

re quegli operatori che già da tempo cercano di percorre vie commerciali diverse, quando propongono un 2000 è: "Chi, Commodore? Quella del C64? Ma un computer che funziona non può costare così poco!". Se l'obiettivo di tanti anni è stato vendere milioni di computer in modo che ogni famiglia (o quasi) avesse un Commodore in casa, quello odierno è vendere al mondo professionale.

Non c'è dubbio che la nuova strategia sia stata dettata dai risultati di vendita che Commodore ha raggiunto in Germania con la linea di PC e, ultimamente anche con i 2000. Purtroppo la situazione tedesca è molto diversa da quella che si può trovare in altri paesi (in Italia, almeno), dove Commodore resta la ditta del C64, abituata a combattere nelle vetrine dei venditori di elettrodomestici con tostapane e forni a micro onde. I prodotti per poter riuscire in questa impresa in effetti esistono, o comunque esisteranno tra breve (vedi Unix per Amiga), ma la strada sarebbe stata molto più semplice da percorrere se si fosse iniziato quando Amiga venne presentato. In quel momento era una macchina con un'ottima grafica, multitasking e a basso prezzo. Ora molti hanno imparato la lezione e le stesse cose, o addirittura migliori, con software adeguato e funzionante già da tempo. Spero di sbagliarmi, poiché ritengo che una professionalizzazione di Commodore avrebbe effetti benefici per tutti gli utenti Amiga, ma ritengo dovrà passare molta acqua sotto i ponti prima che ciò possa avvenire.

La conferenza, a parte una sessione iniziale a cui sono stati invitati tutti i partecipanti, era divisa in una serie di sessioni riguardanti argomenti diversi che si tenevano in sale riservate dell'Intercontinental Hotel. I temi trattati sono spaziati dallo hardware al software, dal modo di utilizzare una certa libreria fino alle indicazioni su come impostare una strategia di marketing, eventualmente favorita da Commodore, per poter vendere i propri prodotti.

Ai partecipanti veniva distribuito un set di dischetti e di documentazione. I primi contenevano l'ultima versione della libreria Janus più tutti i listati degli esempi che si sarebbero trattati nelle sessioni della conferenza. La documentazione era invece composta da un raccoglitore contenente il materiale cartaceo di approfondimento relativo alle diverse sessioni. Molto del materiale è stato direttamente riciclato dalla conferenza tenuta a Washington nell'anno scorso, mentre non si sono viste beta release di future versioni del KickStart: segno che l'1.4 non è proprio dietro l'angolo oppure che non si trattava della sede adatta per distribuirlo.

Nelle ore serali e notturne era aperta una computer room piena di Amiga variamente configurati a disposizione dei visitatori. Si è trattata della sede forse più frequentata e, per certi versi, più interessante. Era infatti possibile scambiare quattro chiacchiere con altri sviluppatori europei o direttamente con personale Commodore, per trovare la soluzione a un problema che magari assillava da qualche mese. Non bisogna inoltre dimenticare che era presente l'intera collezione di Fish Disk, letteralmente presa d'assalto da tutti i presenti. I "Fish", come vengono ormai chiamati, sono infatti considerati da tutti gli sviluppatori come il mezzo (superiore per molti versi alla "Bibbia" ufficiale) tramite il quale si possono ottenere il maggior numero di informazioni ed esempi.

Le sessioni

La giornata tipo dello sviluppatore iniziava alle 9,00, orario di inizio delle prime sessioni mattutine. Alle 10,45 ci si poteva spostare in un'altra sala per ascoltare la descrizione di qualcos'altro, mentre nel pomeriggio non si potevano mancare gli appuntamenti delle 14,00, delle 15,30 e delle 17,00. Poi serata libera e nottata in... computer room o in giro per Francoforte.

Le sessioni più interessanti (a mio modo di vedere), sono state quelle dedicate allo studio del Copper e del Blitter, ai Transputer e allo Unix per Amiga (anche se mancava proprio il diretto interessato: lo Unix). Per quanto riguarda i Transputer avremo sul prossimo numero un articolo che esaminerà proprio le problematiche connesse con l'uso di questo processori ideati per elaborazioni in parallelo. Tanto per dare un'idea degli argomenti trattati, citerò in ordine sparso i titoli di altre sessioni previste durante la conferenza: progettazione di periferiche per il 500, IFF, Mercati per le applicazioni Amiga, software per risoluzioni video maggiori, programmazione dei suoni, programmazione per gli Amiga a 32 bit, Kickstart 1.3 e auto-boot, trucchi e consigli per il C, scrittura di driver per stampanti, gli slot del 2000, creazione di librerie, programmazione in linguaggio assembly, handle e file system, BridgeBoard e libreria Janus.

Le novità

Le novità in pratica... non sono esistite, nel senso che non è stato possibile vedere nulla di nuovo, se si eccettua l'aver visto la possibilità di indirizzare un Megabyte di chip RAM da parte dell'ECS. ECS sta per Enhanced Chip Set: il gruppo di integrati che, rimpiazzando i fratelli minori, possono essere installati negli Amiga 2000 e 500 per togliere il flickering e indirizzare una maggiore quantità di chip RAM. Invece di vedere all'opera qualche novità, è stato possibile avere informazioni su eventuali novità future. I tempi di introduzione e la disponibilità di quanto indicato non è però stato confermato e anzi, come Gail Wellington (rappresentante del managment Commodore d'oltre Oceano) ha invitato i giornalisti a precisare, non è nemmeno garantito che venga realizzato. Tutto ciò è stato però utile per comprendere in quali direzioni si vogli dirigere Commodore con la linea Amiga.

Come abbiamo già detto, lo sforzo maggiore di Commodore è incentrato sulla professionalizzazione di Amiga e della sua immagine. Logico quindi attendersi che la maggior parte delle novità siano da attendersi in futuro per l'Amiga 2000 e non per il 500. Il 2000, attendendo il 3000, dovrebbe essere la macchina di punta per la penetrazione in mercati nuovi e vitali diversi dal settore consumer. Il 500 rimarrà sempre come home computer e usufruirà delle possibili ricadute tecnologiche provenienti dal 2000. Un tipico esempio di ciò è l'unità A590, un hard disk autoconfigurante e autoboot, cioè in grado di far partire il Workbench, ideato per il 500, che è stato creato dopo le esperienze maturate con il fratello maggiore A2090A per il 2000. In futuro potrebbero esserci altri esempi del genere e non è quindi assolutamente il caso di considerare il 500 come una macchina già finita. Probabilmente non si assisterà all'introduzione di nuovi modelli Amiga di fascia bassa (gli sforzi sono concentrati verso l'alto) ma questo non vuol dire che non ven-

gano venduti in futuro milioni di 500. In fondo il C64, quando aveva l'età che ha oggi il 500, si poteva considerare appena nato, quindi il bello dovrebbe cominciare proprio ora.

Ma torniamo ai prodotti.
L'A2286 (BridgeBoard AT) è già in vendita in Italia: si tratta di un buon prodotto, molto più di quanto non fosse la prima BridgeBoard. Funziona in modo splendido se ci si accontenta della visualizzazione che mette a disposizione. Rispetto ai PC AT veri e propri deve cedere diversi punti, ma mette a disposizione, per chi sa sfruttarle, funzionalità davvero uniche.

L'A2620 è una scheda per Amiga 2000 che mette a disposizione un 68020, un 68881 e un 68851. In AmigaDOS non si guadagna un granché in velocità, tranne che con applicativi particolari, ma la scheda è stata espressamente progettata per funzionare con il sistema operativo Unix. Recentemente siamo stati invitati in Commodore per vedere proprio questa scheda all'opera con Unix: al momento non si può commentare. Unix non sfrutta, come potrebbe, nessuna delle possibilità grafiche di Amiga in quanto l'I/O su video è solo di tipo testo. E' possibile aprire più sessioni contemporaneamente (massimo 8) ma restano orientate al trattamento di caratteri. Non si conoscono prezzi né disponibilità, non si sa se e quale sistema di windowing verrà implementato, non si conosce, al momento, se sarà possibile un collegamento in rete. Non è quindi un prodotto reale in grado di competere con avversari già pronti ai nastri di partenza. Si trova nello stato in cui A/UX (lo Unix dell'Apple Macintosh) si trovava due anni fa. A risentirci su questa stessa lunghezza d'onda per eventuali aggiornamenti.

Il famigerato monitor monocromatico ad alta risoluzione A2024 è ancora un sogno. Sicuramente verrà introdotto ma sarà comunque necessario attendere la versione 1.4 del software di sistema. Ricordiamo che, nel formato PAL, mette a disposizione tre risoluzioni: 704 X 256, 704 X 512, 1008 X 1024 tutte non interlacciate. Per chi si occupa di DTP con Amiga, questo monitor sarebbe un vero regalo di Natale, visto che elimina il problema del flickering.

Tra le schede per Amiga 2000 ne sono previste due apposta per applicazioni particolari. A2058 è la sigla di una espansione di memoria da 2 a 8 Mbyte che usa chip da 1 Mbit. L'altra (A2232) è una scheda contenente sette porte seriali funzionanti sia con AmigaDOS che con Unix. Una sessione della conferenza era dedicata all'analisi delle problematiche concernenti l'utilizzo di più porte seriali in un sistema Amiga: una prova che la scheda si farà.

Sul fronte di dischi fissi auto-boot bisogna segnalare la presenza di tre prodotti: A2090A, A2090B e A590. Il primo è una evoluzione del precedente controller per Amiga 2000 A2090: permette l'auto-boot e l'utilizzo di dischi ST506 (quelli dei PC) e SCSI. L'A2090B è una piccola scheda che rende autoboot anche le vecchie unità A2090. L'A590 è ideato per 500 e permette di aggiungere anche fino a un massimo di 2 Mbyte di RAM; si attacca alla porta d'espansione ed è dotato di un alimentatore proprio (senza interruttori d'accensione!) e di una ventola.

Per quanto riguarda il desktop video, si sono avute indicazioni interessanti: Commodore ritiene di aver creato il mercato e non se lo lascerà sfuggire. Questo almeno il senso di due prodotti per il 2000: A2300 e A2350. Il primo è un genlock interno, il secondo dovrebbe essere un adattatore video professionale che realizza le funzioni di un genlock, un digitalizzatore e un frame grabber.

ECS

L'Enhanced Chip Set, annunciato come piano futuro fin dall'introduzione dei modelli 2000 e 500 è stato lungamente atteso da molti utenti. L'attesa dovrà durare ancora un po' visto che sarà necessario l'utilizzo della versione 1.4 del software di sistema. Non porterà comunque rivoluzioni sostanziali, anche perché il software non scritto in maniera corretta (più di quanto si creda) non potrà girare sfruttando le nuove possibilità: sarà necessario attendere una nuova generazione di applicazioni. Le qualità migliori dell'ECS sono la possibilità di indirizzare fino a 2 Mbyte di chip RAM (1 sui modelli attuali) e di mettere a disposizione una risoluzione di 640 X 400 pixel non interlacciata, ma solo con 4 colori estratti da una tavolozza di 64 (peccato!).

Si è parlato più volte della versione 1.4 del software di sistema, vediamo come dovrebbe essere. Innanzitutto avrà il supporto dell'ECS e del monitor monocromatico ad alta risoluzione. Dovrebbe permettere la scalatura automatica dei Bitmap e alcune modifiche ai gadget di sistema. Ci saranno modifiche, speriamo sostanziali, a Preferences e al Workbench, nonché miglioramenti al serial e al parallel device. Ci sarà il supporto dei font a colori (vedi articolo di Betty Clay in questo stesso numero) e per un trattamento del testo più veloce. Chissà che non si riesca a produrre un arcade sullo stile del C64, quando si usavano caratteri predefiniti a colori? Le modifiche maggiori riguarderanno, come c'era da aspettarsi, l'AmigaDOS, già migliorato con la versione 1.3 ma ancora decisamente indietro rispetto alle altri parti del software di sistema. L'FFS (Fast File System) dovrebbe diventare standard anche per i dischetti, esisterà un file requester standard (niente più schifezza come quello dell'AmigaBASIC), una shell ancora migliorata anche nel linguaggio e una maggiore e migliore scelta di comandi. Viene anche annunciata un'espansione della libreria DOS; la speranza di tutti è che si tratti di un qualcosa di simile ad ARP library.

Conclusioni

Gail Wellington, in conclusione della conferenza, ha detto che gli sviluppatori europei hanno posto questioni e domande più intelligenti e pertinenti di quanto non abbiano fatto gli americani in situazioni analoghe. O si tratta di pura demagogia, o è la dimostrazione della vitalità della vecchia Europa che (non dimentichiamo che la prima conferenza europea si è svolta prima che l'Amiga fosse in vendita), ha avuto solo ora la possibilità di confrontarsi con la Commodore. I dati di vendita di Amiga resi noti durante la conferenza, parlano di un milione di Amiga venduti di cui il 70% in Europa. L'Italia non è l'unico paese europeo in cui manca un piano di supporto agli sviluppatori, visti i dati di vendita sarebbe meglio che Commodore concentrasse i propri sforzi dove il mercato è più vitale e offre più opportunità.

I cambiamenti nei font

Finalmente i font a colori

di Betty Clay

Betty Clay è un'insegnante di matematica ("Ritirata finalmente!", dice) per professione e una studiosa di computer della Commodore per vocazione. Betty può essere raggiunta via Compuserve (74145,657), QuantumLink (bjc), o via posta al 1322 South Oak Street, Arlington, Texas, 76010.

Le versioni 1.3 e 1.4 del sistema operativo comporteranno cambiamenti per molti aspetti in Amiga. Tra le novità ci saranno diversi miglioramenti per quanto riguarda l'impiego dei font. Ovviamente, i dati staranno ancora in bitmap e ugualmente ci saranno le tavole CharLoc, CharSpace e affini; insomma, i font attuali funzioneranno ancora. Ad ogni modo, sono in cantiere miglioramenti piuttosto sostanziali e sarebbe bene che i programmatori ne siano al corrente e scrivano programmi che possano farne uso.

Che cosa cambierà

Una delle innovazioni più importanti consiste nel fatto che l'utente non sarà più costretto ad usare Topaz 8 o Topaz 9 come font di default. E' vero che il programma di pubblico dominio SETFONT ci consente già adesso di usare un font di default diverso da Topaz, ma la dimensione rimane comunque predeterminata. Con l'arrivo dei monitor ad alta risoluzione, restrizioni simili non potranno più essere accettate.

Finora il programmatore poteva far affidamento sul fatto che i font fossero a larghezza costante. Le nuove release del software consentiranno di scegliere un font proporzionale come default. Bisogna allora che i programmatori tengano conto del fatto che il calcolo dello spazio occupato da un font proporzionale impiega una tecnica diversa da quella solitamente usata per font a larghezza fissa.

Un altro cambiamento della versione 1.3 è l'introduzione dei ColorFonts. Il software di sistema che consente il loro uso è stato sviluppato in collaborazione con la Interactive Software, la casa che ha pubblicato Calligrapher. La struttura ColorFont sarà resa totalmente operativa solo nella release 1.4, ma già nella 1.3 sarà possibile l'uso di font a più colori, però, con la limitazione di un range ben determinato di colori. Si potrà, inoltre, sostituire la color map di un'applicazione con quella prevista dal disegnatore del font e trasferire un colore del font nella variabile APen del RastPort. Tra le possibilità che verranno introdotte successivamente ci sono il "remapping" dei piani di CharData con PlanePick e PlaneOnOff e il "remapping" dei colori di CharData per un font a scala di grigi. Quando la struttura ColorFont sarà pienamente operativa (cioè nella 1.4), essa farà parte della graphics.library.

Se vi è capitato di usare font presi da una delle prime versioni di Calligrapher, potreste aver notato che alcuni di essi, in pratica non potevano essere utilizzati per l'elevata richiesta di memoria e potreste allora essere prevenuti all'uso dei ColorFonts. Nelle versioni più recenti di Calligrapher, il bug responsabile dell'uso eccessivo della memoria è stato corretto e comunque esiste un programma di pubblico dominio, FONTREP, che ripara intere directory di font di Calligrapher con un solo comando. Il problema era causato dal fatto che il campo del nome del font non era ben allineato e il nome del font veniva messo nel posto sbagliato, cosicché il font veniva caricato ogni volta che serviva senza prima liberare la memoria già occupata. La nuova diskfont.library corregge automaticamente questi font al momento di caricarli, assicurando così che font a colori di differenti versioni funzionino ugualmente bene.

Questi ora esposti sono i tre cambiamenti principali. La disponibilità di un gran numero di dischi pieni zeppi di font, l'introduzione dei monitor ad alta risoluzione e la presenza sul mercato di svariati programmi che fanno uso dei font a colori hanno reso tali innovazioni necessarie e auspicabili. Rimane allora da esaminare che cosa comporteranno a livello del programmatore.

Attenzione!!!

Ci sono diverse cose che il programmatore deve fare per assicurarsi che i programmi continuino a funzionare correttamente e mantengano un "bell'aspetto" in queste nuove circostanze. Ecco le considerazioni di maggior importanza:

1. I programmi devono trattare senza problemi grandi quantità di font.

Molti programmi attualmente in circolazione sono soggetti a questo tipo di problema. Probabilmente il più noto e DPaint. Va inevitabilmente in guru quando deve trattare una lunga lista di font e questo è un difetto molto grave. La possibilità di im-

piegare un'ampia varietà di font è importante in svariate applicazioni di DPaint, ma il programma può accedere a un numero limitato di font e non offre alcun modo per leggere una diversa directory Fonts:.

Un caso perfino peggiore è rappresentato da DeLuxePrint. Questo programma consente l'uso solo dei font disponibili sul Workbench originale. È triste che due programmi così utili siano rovinati da simili difetti; a loro discolpa c'è comunque da osservare che furono tra i primi programmi sviluppati per Amiga. Oggi, però, un programmatore non può più permettersi di ripetere questi errori.

2. I programmi devono poter trattare font con diverse dimensioni.

Quando un programmatore prova il suo programma con l'attuale font di default, probabilmente gli apparirà magnifico. Quando il suo cliente usa quel programma su un monitor ad alta risoluzione come l'A2024, potrebbe invece risultare del tutto illeggibile. Il nuovo software dovrà sempre permettere che l'utente scelga il font che più preferisce; se il programmatore non pone un po' di attenzione a questo riguardo, il font scelto dall'utente potrebbe, ad esempio, non essere adeguato per lo spazio che il programmatore ha riservato per un certo testo. Con un po' di cura, invece, il software può essere reso flessibile senza peraltro sacrificare nessuno degli effetti desiderati.

3. Fin dai primissimi tempi, la Commodore-Amiga ha richiesto che i programmatori seguissero le regole esposte nel ROM Kernel Manual.

Quando tali convenzioni vengono osservate, le nuove release del software di sistema provvedono loro stesse ad assicurare che i programmi prodotti precedentemente continuino a funzionare. Certi programmatori si divertono a trovare espedienti per aggirare le regole; capita allora che i loro programmi si trovino a mal partito quando il software di sistema cambia. Questo riguarda i font come molti altri aspetti dell'Amiga.

Dove trovare l'informazione

La maggior parte dei testi è posta sullo schermo tramite Intuition, per cui è opportuno sapere dove Intuition cerchi i font che deve usare. Generalmente, il testo che si vuole porre sul schermo è contenuto in una struttura IntuiText, nella quale il campo ITextFont (che è un puntatore a una struttura TextAttr) specifica il font da usare. Nel caso, invece, in cui ciò non sia specificato, Intuition userà il font che trova in uno dei tre posti seguenti:

```
Screen->Font,
Screen->RastPort.Font,
Screen->BarLayer-rp-Font
```

a seconda dell'impiego che ne deve fare. Quando viene aperto uno schermo, questi tre font, in realtà, si riferiscono tutti ad un medesimo font e cioè quello indicato nella struttura NewScreen, se è stato possibile trovarlo e aprirlo; in caso contrario, invece, viene impiegato il font di default. È possibile cambiare ognuno di questi tre font mentre il programma gira, ma se uno viene cambiato, allora devono esserlo tutti e tre.

La struttura IntuiText viene usata per produrre la maggior parte dei testi che appaiono sullo schermo; i testi nei gadget, i menu, i titoli delle window e degli screen, i testi dei "requester" e degli "alert" sfruttano tutti questo sistema. È dunque facile rendersi conto che cambiare i font che Intuition usa nelle varie circostanze può causare notevoli problemi. Si può, ad esempio, osservare a tal riguardo che l'altezza delle "title bar" nelle window e negli screen viene determinata dall'altezza del font che vi viene impiegato. Se si cambiano i font usati per quello scopo mentre il programma gira, è molto probabile che si verifichino inconvenienti piuttosto notevoli.

La cosa forse più importante da ricordare su Intuition e i font è che Intuition non carica mai un font da disco. Se il font desiderato non è già in memoria e pronto per l'uso, Intuition usa il font di default. Se si desidera impiegare un font residente su disco con Intuition, è allora necessario che tale font venga aperto prima dell'uso con Intuition e chiuso al termine del programma.

Calcolo delle dimensioni

Dal momento che gli utenti dovranno essere lasciati liberi di scegliere il font che soddisfa il loro gusto o i loro requisiti e poiché il programmatore non può sapere prima le caratteristiche dei font che saranno usati, bisogna trovare un modo per far sì che il software si adatti a qualsiasi font.

Una maniera molto ovvia è quella di specificare un font particolare e di assicurarsi che sia disponibile al programma quando richiesto. L'utente, però, potrebbe non gradire quel certo font; magari sul suo schermo potrebbe risultare del tutto illeggibile. D'altro canto la predeterminazione consente al programmatore di assicurarsi che ci sia sufficiente spazio nei menu, nei gadget, ecc. per i testi che vi devono apparire.

È di gran lunga più elegante, però, lasciar libero l'utente di scegliere il font che più gli aggrada e far sì che il software trovi le caratteristiche di tale font e conseguentemente vi si adatti; infatti, l'altezza di un font è precisata nelle strutture TextFont e RastPort, mentre la lunghezza in pixel di una stringa può essere valutata chiamando la funzione TextLength() o la funzione IntuiTextLength().

Si può perfino creare un RastPort per il solo scopo di fare i calcoli sui testi. Prima di aprire uno screen custom, il programmatore può chiamare InitRastPort() e SetFont() e quindi calcolare senza problemi di alcun tipo le dimensioni delle "title bar" e di tutto quello che dovrà apparire sullo schermo. Questo metodo funzionerà ugualmente bene per un font scelto dall'utente, per uno predeterminato dal programmatore o per quello di default. Le informazioni sul font di default possono essere ottenute riferendosi a GfxBase->DefaultFont.

Dobbiamo poi pensare che nel futuro ci saranno ulteriori cambiamenti. Verranno aggiunte nuove strutture a quelle già esistenti che renderanno più facile il calcolo dello spazio richiesto usando un font imprecisato. Molto probabilmente avremo ben presto file separati per il grassetto e per il corsivo e metodi

soddisfacenti per "scalare" font preesistenti con una distorsione minore di quella che si può osservare oggi. Inoltre bisogna ricordare che Amiga è venduto in molte nazioni; affinché un programma possa essere usato nei diversi paesi, i font che usiamo devono contenere i caratteri usati nelle diverse lingue.

La situazione dunque si sta evolvendo e continuerà sicuramente ad evolversi nel futuro. Amiga non può che migliorare.

I file ".font"

Nel primo numero di Transactor abbiamo spiegato la struttura dei file che specificano il contenuto del font ovvero dei cosiddetti file ".font". Basta ricordare che essi contengono un identificatore ($0x0F00), il numero di componenti della famiglia di font in questione, 256 byte in cui è precisato il "path" necessario per accedere al vero e proprio font e infine l'altezza, lo stile e i flag del font. Se ci sono font di più dimensioni o di diversi stili nella stessa famiglia, tutte le informazioni a esclusione dell'identificatore sono ripetute per ogni font. Lo scopo di questi file, che risultano comunque molto corti, è di rendere più veloce la localizzazione del font desiderato.

Solo recentemente qualcuno ha pensato di sfruttare in modo proficuo i file ".font". Stephen Vermeulen, l'autore di Express Paint, ha ultimamente realizzato un piccolo programma che ha chiamato REN e che costituisce una delle più comode utility per i font finora apparse. Usando REN e alcuni dei font che sono riuscita a collezionare, ho preparato di recente due dischi di font che contengono solamente i veri e propri dati, mentre i file ".font", che occupano poco spazio rimangono nel mio disco di boot. Sia PageSetter che NotePad non hanno avuto alcun problema per localizzare i font che ho richiesto; DPaint, invece, è andato dritto dritto in "guru".

Questo è stato possibile solo perché Stephen ha finalmente pensato di impiegare quei 256 byte per il loro scopo dichiarato. A REN dovete solo dire dove volete mettere i dati di un font e il programma scriverà il path specificato nella zona riservata del file ".font" per ogni dimensione e per ogni stile. Quando richiedete un certo font e il disco su cui stanno i dati non è disponibile, verrà fuori il solito requester standard che vi dirà "Please place Volume xxxxxxxx in any drive", cioè "Inserite il disco xxxxxxxx in un qualsiasi drive". Allora inserite il disco appropriato e le cose andranno nella stessa maniera in cui andrebbero se i font affollassero, invece, il vostro boot disk!

Con il permesso di Stephen, ecco il programma:

```
/*****************************************************
  ren.c

  Questo programma permette di cambiare nome alla
  directory che contiene i file dei font.

  Copyright 1988 By Stephen Vermeulen

  Questo programma puo' essere distribuito
  liberamente finche' il messaggio di Copyright
  rimane intatto e finche' e' distribuito con la
  documentazione fornita.

  sintassi:

      ren questofont.font path
*****************************************************/

#include <stdio.h>

char newname[256], oldname[256];

main(argc, argv)
int argc;
char *argv[];
{
  FILE *font;
  long j, pos;
  short i, n;

  if (argc != 3)
  {
    printf("ren filename.font nuovopath\n");
    printf("Copyright 1988 By Stephen Vermeulen\n");
    printf("3635 Utah Dr. N.W.,  Calgary, Alberta, \
CANADA, T2N 4A6\n");
  }
  else
  {
    font = fopen(argv[1], "r+");
    if (font)
    {
/*************************************************
 ora determiniamo quante dimensioni ha questo font
 *************************************************/
      fread(&n, 2, 1, font);  /** butta via **/
      fread(&n, 2, 1, font);  /** dimensioni **/
      printf("Il font %s ha %d dimensioni\n",
              argv[1], n);
      for (i = 0; i < n; ++i)
      {
        /** per ogni dimensione cambiamo nome...
         **/

        fseek(font, 4L + 260L * i, 0);
        fread(oldname, 256, 1, font);
        printf("Il nome era: %s ", oldname);
        for (j = 255; j > 0; --j)
          if (oldname[j] == '/') break;
        strcpy(newname, argv[2]);
        strcat(newname, oldname + j);
        fseek(font, 4L + 260L * i, 0);
        printf("ora e': %s\n", newname);
        fwrite(newname, 256, 1, font);
      }
      fclose(font);
    }
  }
}
```

Come richiamare il CLI dall'AmigaBASIC

Il Basic come il C

di Jim Butterfield

Jim Butterfield è uno scrittore, programmatore e conferenziere che vive a Toronto il cui legame con i microcomputer inizia con il 1K KIM-1 agli albori della microinformatica casalinga. L'abilità di Jim di comunicare la sua conoscenza enciclopedica dei prodotti CBM attraverso articoli, libri, conferenze e programmi televisivi ha fatto si che il suo nome diventasse familiare tra gli utenti Commodore nel mondo intero.

L'AmigaBASIC è un linguaggio veloce e potente, fornito assieme ad ogni computer Amiga, che possiede un ampio vocabolario e che può adempiere a molte funzioni. Nonostante ciò a volte si vorrebbe poter fare qualcosa che risulta più facile o più naturale in CLI - l'interprete dei comandi dell'AmigaDOS. Questo articolo intende spiegare come poter realizzare ciò.

Quasi tutti i programmi che sono eseguibili sull'Amiga possono essere lanciati da CLI, in modo che riuscendo ad accedere al CLI dall'AmigaBASIC si può avere l'intera macchina sulla punta delle dita. Prima di arrivare a questo bisogna però imparare alcune regole fondamentali.

Indagare sulla libreria

Per collegarsi con la potenza del sistema operativo dell'Amiga, bisogna mettersi in comunicazione con le librerie di sistema. Alcune di queste risiedono nella ROM del Kickstart, altre in RAM e altre ancora sul disco. In ogni caso non bisogna preoccuparsi di dove siano ma quello di cui si ha bisogno è di avere insieme tre elementi.

Il primo è l'istruzione LIBRARY del BASIC. Quando si usa LIBRARY seguita dal nome di una libreria, il BASIC effettua il collegamento con i processi interni dell'Amiga. Per i nostri scopi, si farà uso della libreria del DOS (Disk Operating System) e il comando che verrà usato sarà:

```
LIBRARY "dos.library"
```

Accostandosi al sistema operativo dell'Amiga bisogna sviluppare una nuova abitudine: la consapevolezza della differenza tra caratteri maiuscoli e minuscoli. Nell'AmigaBASIC *normale* si possono mischiare minuscole e maiuscole come si vuole e non importa se si scrive "print" oppure "PRINT" o addirittura "PrInT". Il BASIC capisce quello che si intendeva inserire. In modo analogo, si può chiamare una variabile "subtotal" oppure "SubTotal" e in entrambi i casi verrà considerata la stessa variabile senza possibilità di ambiguità. Anche i nomi dei file sono esenti da questo tipo di problema, mentre il sistema operativo ne risente pesantemente e quindi bisogna usare le minuscole e le maiuscole nel modo in cui vengono riportate nei sorgenti.

Ci si collega a una libreria attraverso la funzione LIBRARY, mentre quando si terminano le attività con essa bisogna scollegarsi usando l'istruzione LIBRARY CLOSE. Possono essere attive più di una libreria alla volta. Per fare sì che il comando LIBRARY funzioni correttamente si ha bisogno di un altro elemento: il file *.bmap*. Ne parleremo in dettaglio più avanti, comunque se non si possiede il file *.bmap* appropriato bisogna crearne uno.

Se si vogliono chiamare solamente dei sottoprogrammi della libreria, basta il comando LIBRARY con associato il file *.bmap* appropriato, mentre se si vogliono chiamare delle funzioni bisogna utilizzare un altro comando: DECLARE FUNCTION seguito da un nome di funzione e dalla parola LIBRARY.

Sottoprogrammi contro funzioni

Un sottoprogramma è molto simile a una subroutine; lo si può chiamare e lui realizza qualcosa. Lo si può associare a una chiamata del tipo "GOSUB DingDong"... si salta da qualche parte, viene eseguito del codice e quindi si ritorna al punto successivo alla chiamata. In modo più accurato, si potrebbe usare "CALL xClose"... ma l'idea resta la stessa.

Una funzione, invece, calcola un valore. Per esempio, se si vuole calcolare la radice quadrata di un numero bisogna usare la funzione BASIC SQR(). Non si usa mai una funzione da sola, ma è sempre parte di un'espressione del tipo:

```
PRINT SQR(16) + SQR(25)
```

Una funzione di libreria deve essere dichiarata esplicitamente prima di poterla usare, con DECLARE FUNCTION. Lo scopo più evidente di una dichiarazione di questo tipo è quello di identificare il tipo del valore che verrà restituito dalla funzione. Il tipo è quasi sempre un long integer, specificato inserendo il carattere '&' dopo il nome della funzione. Quindi, prima di usare una funzione chiamata xOpen, si dovrà inserire:

```
DECLARE FUNCTION xOpen&() LIBRARY
```

Si potrebbe dedurre, basandosi su di una passata esperienza con il BASIC, che i sottoprogrammi servono a realizzare delle cose mentre le funzioni sono dedicate al calcolo di valori. In realtà non è così; le funzioni vengono usate con l'Amiga per fare quasi qualsiasi cosa. Vediamone il funzionamento.

Supponiamo, per esempio, che si debba aprire un canale di uscita. Sembrerebbe che sia necessario utilizzare un sottoprogramma. In effetti è una funzione che potrebbe essere chiamata con:

```
h& = xOpen&(argomenti)
```

Quando si chiama la funzione **xOpen&** (attenzione alle lettere minuscole e maiuscole), essa apre un canale come era stato richiesto, e quindi ritorna una identità del canale (chiamato handle) che si può usare ogni volta che ci sia la necessità di far riferimento a quel canale. Se, per qualche motivo, non potesse aprire il canale ritornerebbe il valore zero. Si potrebbe quindi aggiungere all'istruzione precedente:

```
IF h&=0 THEN PRINT "Non posso aprire il
      canale..":STOP
```

Se l' handle è diverso da zero il canale è stato aperto con successo. L' handle, in sostanza, contiene un indirizzo che verrà passato a qualsiasi altra attività che abbia bisogno di usare quel canale. In ogni caso non bisogna preoccuparsi di esso, basta seguire le regole e tutto funzionerà nel migliore dei modi.

Il file *.bmap*

Se, dall'ambiente CLI, si lancia il comando: "DIR LIBS:" si vedranno diversi elementi memorizzati in questa directory. Si potrebbe osservare anche un file chiamato dos.bmap, anche se probabilmente ciò non è vero. In caso esista questo file si può saltare direttamente al prossimo paragrafo, in caso contrario bisogna crearne uno.

L'appendice F del manuale dell'AmigaBASIC contiene molto materiale oscuro al riguardo dei file *.bmap*. Parleremo in dettaglio di questo nella parte 3. Per ora, è importante solo l'ultima frase la quale dice: il modo migliore per ottenere un file *.bmap* è quello di tirare fuori il vostro disco di EXTRAS (quello che contiene l'AmigaBASIC), selezionare la sezione BasicDemos e far partire il programma "ConvertFd".

Se si vuole fare così, bisogna ricordare che il nome del file che si vuole convertire si chiama DOS. Dopo la conversione vi ritroverete con un file chiamato dos.bmap che però probabilmente è già presente sul disco, nella libreria BasicDemos.

In ogni caso, la prossima cosa da fare è di copiarlo in modo che sia presente anche nella directory LIBS: dove può vivere felicemente per sempre. La copia si può effettuare da CLI con il comando:

```
COPY EXTRAS:BasicDemos/dos.bmap LIBS:
```

Se tutto ciò sembrasse troppo complicato, potreste digitare il Programma 1, Bmap_Maker, che crea una versione ridotta del file dos.bmap. Mandando in esecuzione il programma verrà creato il file e anche un secondo file chiamato dos.bmap.info, non necessario e che quindi può essere cancellato.

Come arrivare al CLI

Ora che il dos.bmap è installato in modo sicuro nella libreria LIBS: si può passare ad analizzare l'espediente usato per giungere al nostro scopo.

Il comando Execute della libreria DOS esegue ogni comando CLI che gli viene passato e questo è il punto dove il BASIC, in realtà, ha accesso all'intera macchina. Si farà in modo che Execute esegua il comando SORT, in modo da permettere di effettuare velocemente l'ordinamento di un file.

Il Programma 2, SortCli, permette di inserire dei nomi che verranno passati, non appena ricevuti, al RAMdisk in un file chiamato TEMP0. Una volta terminata l'operazione di inserimento, il file viene chiuso e inizia la parte interessante.

Viene aperta la LIBRARY e viene definita la funzione Execute&. Da notare che questa è una funzione, e non un sottoprogramma e quindi ritorna un valore (un flag booleano relativo al successo dell'operazione).

A questo punto viene eseguito il comando SORT TEMP0 TEMP1 e vengono terminate le attività legate alla libreria. Resta solamente, per nostra soddisfazione, da ripescare i valori ordinati in TEMP1 e la loro stampa.

Avvertimento

Questa versione del programma funziona solamente se si chiama l'AmigaBASIC dal CLI. Bisogna perciò, entrare in ambiente CLI ed eseguire "AmigaBASIC" oppure "Run AmigaBASIC".

Se si lancia l'AmigaBASIC dal Workbench, selezionando l'icona del programma o l'icona dell'AmigaBASIC, il programma andrà tristemente in crash.

State sintonizzati per sapere perché e come risolvere il problema...

Dettagli del programma

Dovrebbe valere la pena dare un'occhiata al codice del programma SortCli per vedere come funziona la chiamata al CLI. La maggior parte del programma è scritto in *BASIC normale*. Per esempio, c'è del codice all'inizio che permette l'inserimento dei dati che vengono passati immediatamente a un file in

RAM, e c'è del codice alla fine che stampa il contenuto del file ordinato in RAM. Sono comunque scritti in un BASIC lineare.

Il codice che realizza la connessione con il CLI si trova a metà programma. La prima istruzione usata è:

```
DECLARE FUNCTION Execute& LIBRARY
```

Questa linea dice, "Quando uso una funzione chiamata Execute, voglio che il computer sappia che è una funzione di libreria, e che il valore restituito dalla funzione sarà un long integer".

Come è già stato puntualizzato, Execute non è un comando, ma una funzione che ritorna un valore di tipo booleano, cioè segnala un vero o un falso. In questo caso la segnalazione "operazione completata con successo" è associata al vero.

Nel manuale dell'AmigaDOS si possono leggere informazioni aggiuntive relative a tutto ciò. Nella parte intitolata "AmigaDOS Developer's Manual" si trova "Calling AmigaDOS" (Capitolo 2), e questo capitolo fà riferimento alla funzione Execute nella Sezione 2.2 sotto l'intestazione Loading Code.

Il formato è:

```
Success = Execute( commandString, input,
    output )
```

Dopo aver preparato la stringa di comando, posti a zero input e output, e aver chiamato la funzione Execute si riceverà un valore di ritorno relativo al successo dell'operazione. Ma prima di tutto questo, bisogna aprire la libreria con:

```
LIBRARY "dos.library"
```

Ora, per la chiamata effettiva al CLI bisogna usare:

```
x=Execute&(SADD("sort >nil: ram:temp0
ram:temp1"
    +CHR$(0)), 0, 0)
```

La stringa di comando è racchiusa tra virgolette e deve essere seguita da un carattere NULL perché questo è il modo in cui il sistema riconosce la fine di un comando. In questo modo abbiamo fornito l'indirizzo della stringa di comandi e abbiamo posto input e output a zero. La chiamata ad Execute restituirà una segnalazione di "successo" che verrà posta nella variabile x...che non verrà mai usata.

Ora che il lavoro è terminato, bisogna concludere con:

```
LIBRARY CLOSE
```

Da questo punto in poi si continua con il BASIC *normale*.

Dove è l'errore?

Tutto questo è molto efficente, ma perché il programma non funziona nell'ambiente del Workbench? Per trovare una spiegazione bisogna leggere dell'ulteriore documentazione relativa al comando Execute:

"...nella maggior parte dei casi deve essere fornito l' handle del file di output, il quale verrà usato dai comandi del CLI come loro flusso privato di output a meno che non sia stata specificata una redirezione."

Bene, SORT non ha bisogno di un flusso di output perché gli è stato specificato dove mettere il file ordinato. Si legge ancora:

"...Se l' handle del file di output è posto a zero allora viene usata la finestra corrente, che viene normalmente specificata come *. Da notare che i programmi che vengono eseguiti sotto il Workbench, normalmente non hanno una finestra corrente."

Questo è il problema, bisogna avere una finestra di output, che non viene fornita dal Workbench, ma che deve esistere anche se ad essa non viene inviato mai nulla.

Si deve preparare un percorso per l'output, anche se in realtà non verrà mai usato. Ponendo un output a NIL... qualunque cosa passi su di esso verrebbe comunque persa. Questo è un tipo di doppia negazione: non si stà inviando nulla e si butta via ciò che dovrebbe arrivare. Ma, nonostante tutto, è un modo per soddisfare la necessità di un percorso di output.

Apertura del percorso

Nella stessa documentazione si trovano dei dettagli riguardanti la funzione Open la quale, ricorderete, restituisce un valore detto handle, che verrà poi usato per aprire un percorso fino a NIL. Più avanti il percorso verrà chiuso usando il sottoprogramma Close, il quale non essendo una funzione non ha bisogno di essere dichiarato. Il formato per la funzione Open è il seguente:

```
handle = Open( name, AccessMode )
```

Il name è una stringa che verrà gestita allo stesso modo con cui si era operato per la funzione Execute, cioé verrà terminata con uno zero e verrà fornito il suo indirizzo e non la stringa stessa. L'AccessMode prevede solamente tre valori validi: 1004 per aprire un file in lettura e in scrittura; 1005 per aprire un file in lettura; e 1006 per aprire un file in scrittura. Handle è il valore che verrà restituito e che permetterà l'identificazione del file. Sarà un long integer e bisognerà ricordarsi di ciò quando si definirà la funzione.

Ecco come si procederà: Si deve ricordare di usare DEFINE FUNCTION per la funzione Open (risiede nella stessa libreria della Execute). Una volta che la chiamata ad Open restituisce l'handle del file, quest'ultima verrà passata ad Execute. Ora, questa funzione ha associato un percorso di output e quindi funzionerà anche quando verrà chiamata dal Workbench.

Esiste però un piccolo problema: il file dos.bmap non contiene una funzione chiamata Open, ma possiede invece una funzione detta xOpen. La causa di ciò è che OPEN è una parola riservata del BASIC, e quindi l'AmigaBASIC sarebbe confuso dal suo utilizzo. Quindi si userà xOpen (e, più avanti, xClose), senza tenere conto della 'x' quando si legge la documentazione.

Provando a utilizzare il terzo programma, SortWb, ci si accor-

gerà che esso funziona indipendentemente dall'ambiente in cui viene chiamato l'AmigaBASIC.

I file *.bmap*

È ora di dare un'occhiata più approfondita ai file *.bmap*, di fornire qualche accenno al funzionamento dell'Amiga, e di creare un programma che esemplifichi i vari modi con cui si può usare il CLI come supporto all'AmigaBASIC.

Per certi aspetti, la struttura dei file *.bmap* riflette la struttura dell'Amiga stesso. Vale la pena di dare un'occhiata da vicino, e il programma Bmap_Snoop (quarto programma) lo farà per voi.

Struttura delle librerie

L'Amiga contiene un certo numero di strutture dette librerie che rendono possibile l'accesso da parte dei programmatori alle risorse più interne della macchina. Queste librerie contengono un gran quantità di cose interessanti, ma la parte a cui siamo strettamente interessati ora si chiama jump table (tabella dei salti). Una jump table è una collezione di istruzioni 'jump' che indicano al calcolatore di saltare in una certa locazione per svolgere un determinato compito.

Ciascuna libreria ha associata una jump table differente, personalizzata per il proprio compito. La libreria DOS, per esempio, contiene i punti di ingresso per le routine di cui abbiamo già parlato (Open, Close ed Execute) più molte altre come DeleteFile, Read e Lock. Ciascuna di queste funzioni occupa una ben definita posizione all'interno della jump table.

Una libreria ha un *punto di riferimento*, o indirizzo, il quale la localizza, ma che non si trova all'inizio della struttura della libreria stessa. Gli indirizzi al di sopra del punto di riferimento di solito contengono un certo tipo di dati, mentre quelli al di sotto contengono la jump table. Non bisogna quindi sorprendersi se un elemento della jump table è indicato alla locazione -60 nella sua libreria... è il punto dove risiede di solito.

Cosa abbiamo bisogno di conoscere per essere in grado di effettuare il collegamento ? Per prima cosa si deve sapere il nome della libreria (nel nostro esempio era "dos.library"). Ogni libreria contiene il proprio nome; un comando del tipo LIBRARY "dos.library" farà in modo che il BASIC forzi il sistema operativo a cercare una libreria con quel nome e a riportarne la sua posizione. Una libreria può essere spostata, ma non quando qualcuno la sta utilizzando.

Una volta trovata la libreria, grazie al comando LIBRARY, bisogna rintracciare la funzione che si desidera utilizzare: Open, Execute, ecc. Si deve conoscere a quale offset, all'interno della libreria, è posizionata la funzione che interessa. Il file *.bmap* fornisce questo tipo di informazioni.

Il file *.bmap* contiene i nomi degli elementi esistenti in una data libreria e gli offset che sono a loro associati. L'offset è sempre negativo. Esiste un altro pezzo di informazione contenuto nel file, ma verrà trattato più avanti.

Per esempio, la funzione Open è la prima nel file dos.bmap e ad essa è associato un offset di -30, mentre la funzione Execute è all'ultimo posto con un offset di -222 (è una jump table molto grande).

Conoscendo il nome della libreria e avendo un file *.bmap*, si può raggiungere direttamente la zona della jump table che ci interessa. Esiste solamente un altro problema da risolvere: quanti dati si devono passare a una funzione? Alcune funzioni non richiedono dati, mentre altre necessitano di una raccolta di informazioni, che devono essere accuratamente ordinate nei registri del processore quando viene chiamata la funzione. Come può un programma BASIC gestire questi dati?

Nuovamente, i file *.bmap* forniscono la risposta al quesito. Oltre a contenere l'offset per la funzione, il file ha associate informazioni relative a come debbano essere preparati i dati affinché la funzione si comporti in modo corretto. È presente una lista di registri per indicare all'interfaccia che tipo di informazione deve essere fornita e dove deve essere posta. I registri possono essere indicati con una 'D' per i registri di dato, e con una 'A' per i registri di indirizzo, seguiti, in entrambi i casi, da un numero da 0 a 7.

Non bisogna preoccuparsi della funzione svolta dai registri, in quanto se ne occuperà il calcolatore. E' invece utile sapere che un registro può contenere un valore intero grande parecchi milioni, oppure un indirizzo di memoria. Se è necessario usare un numero, è meglio usarlo direttamente; se c'è bisogno di una stringa si deve passare il suo indirizzo e terminarla con un carattere NULL per indicarne la fine.

Il programma Bmap_Snoop

Provate a far eseguire questo programma. Chiederà il nome di qualche file *.bmap* noto; si potrà scegliere tra uno di quelli proposti o inserire il proprio.

Il programma stamperà i nomi delle funzioni presenti nella libreria, gli offset, e i registri che sono necessari. Non esiste nulla di segreto, è solo una semplice tabella, ma noterete che sono presenti un gran numero di funzioni e che ci vuole un mucchio di tempo e molte consultazioni di manuali, per poter apprezzare tutto quello che sono in grado di svolgere.

Il programma DirMake

Riassumendo, ho scritto il programma DirMake (Program 5) per esporre i differenti modi con cui si possono interfacciare il BASIC ed il CLI. Il programma implementa qualcosa che non è molto facile fare in BASIC... crea una nuova directory.

La prima chiamata ad Execute esegue il comando INFO del CLI. L'output viene inviato nel file RAM:temp, ed è usato per individuare i dispositivi correntemente attivi (dischi, RAM ecc.). Sceglietene uno a piacimento.

Nella prima chiamata, il comando del CLI viene specificato direttamente nella linea dell'Execute.

La seconda chiamata, che può essere ripetuta, guarda in una

data directory, usando il comando DIR del CLI, per vedere se all'interno di essa esistono altre subdirectory. Se questo è vero, è permesso scegliene una e questa parte del programma viene ripetuta.

Nella seconda chiamata, la stringa di comando viene formata separatamente e referenziata come una variabile dal comando Execute.

La terza chiamata usa il comando MAKEDIR del CLI per creare la directory desiderata. Per variare, e per dimostrare che si può fare anche così, il comando viene inserito in un file testo che Execute manderà in esecuzione. In questo caso il comando era unico, ma si può far eseguire, se si desidera, tutta una serie di comandi.

Conclusioni

Usando la magia dei comandi LIBRARY e DECLARE FUNCTION e l'elegante file *.bmap*, si può aggiungere una gran varietà di comandi al BASIC.

E facendo ciò, si può riuscire a vedere più a fondo nel funzionamento dell'Amiga.

Listato 1:

```
REM   Bmap-maker       Jim Butterfield

DATA 120,79,112,101,110,0,255,226,2,3
DATA 0,120,67,108,111,115,101,0,255,220
DATA 2,0,69,120,101,99,117,116,101,0
DATA 255,34,2,3,4,0

FOR j=1 TO 36
 READ v
 t=t+v
 m$=m$+CHR$(v)
NEXT j
IF t<>3128 THEN PRINT "ERRORE NEI DATA":END

PRINT "sto generando il file <libs:dos.bmap> ..."
 OPEN "libs:dos.bmap" FOR OUTPUT AS #1
  PRINT#1,m$;
 CLOSE #1
PRINT "file <libs:dos.bmap> scritto"
```

Listato 2:

```
REM - questo programma dimostra come richiamare la
REM - routine EXECUTE dall'Amiga-BASIC

PRINT "Name sorter demo ... Jim Butterfield"

PRINT "Inserite una serie di nomi.."
PRINT "terminanti con una linea bianca."
OPEN "ram:temp0" FOR OUTPUT AS #2
n=1
PRINT "Nome";n;": ";
LINE INPUT n$
WHILE n$"<>"
 PRINT#2,n$
 n=n+1
 PRINT "Nome";n;": ";
 LINE INPUT n$
WEND
CLOSE #2
PRINT
PRINT "Ordinamento dei nomi...";
DECLARE FUNCTION Execute& LIBRARY
  REM --- La prossima linea funziona solo in CLI,
  REM --- non con il WorkBench
  LIBRARY "dos.library"
  x=Execute&(SADD("sort >nil: ram:temp0
                   ram:temp1"+CHR$(0)), 0, 0)
  LIBRARY CLOSE
PRINT "... ordinamento completato."
PRINT "   Ecco i nomi:"
OPEN "RAM:temp1" FOR INPUT AS 1
WHILE NOT EOF(1)
  LINE INPUT #1,a$
  PRINT a$
WEND
CLOSE
KILL "RAM:temp0"
KILL "RAM:temp1"
PRINT
END
```

Listato 3:

```
REM - questo programma dimostra come richiamare la
REM - routine EXECUTE dall'Amiga-BASIC

PRINT "Name sorter demo ... Jim Butterfield"

PRINT "Inserite una serie di nomi.."
PRINT "terminanti con una linea bianca."
OPEN "ram:temp0" FOR OUTPUT AS #2
n=1
PRINT "Nome";n;": ";
LINE INPUT n$
WHILE n$"<>"
 PRINT#2,n$
 n=n+1
 PRINT "Nome";n;": ";
 LINE INPUT n$
WEND
CLOSE #2
PRINT
PRINT "Ordinamento dei nomi...";
DECLARE FUNCTION xOpen& LIBRARY
DECLARE FUNCTION xClose& LIBRARY
DECLARE FUNCTION Execute& LIBRARY
 '  Questo funziona sia con il CLI che con il
 '  Workbench perche' si crea il proprio output NIL:
```

```
 LIBRARY "dos.library"
 z$=CHR$(0)
 handle&=xOpen&(SADD("NIL:"+z$),SADD("MODE_NEWFILE"+
                                      z$))
 IF handle&<>0 THEN
  x=Execute&(SADD("sort ram:temp0 ram:temp1"+z$), 0,
                  handle&)
  xClose(handle&)
 ELSE
  PRINT "Non posso gestire l'i/o"
 END IF
 LIBRARY CLOSE
PRINT "... ordinamento completato."
PRINT "   Ecco i nomi:"
OPEN "RAM:temp1" FOR INPUT AS 1
WHILE NOT EOF(1)
  LINE INPUT #1,a$
  PRINT a$
WEND
CLOSE
KILL "RAM:temp0"
KILL "RAM:temp1"
PRINT
END
```

Listato 4:

```
PRINT " BMAP Looker   -   Jim Butterfield"
PRINT
DATA diskfont,dos,graphics,exec,altro,fine
READ f$
f=0
WHILE f$<>"fine"
 f=f+1
 IF f$="altro" THEN
  PRINT f;" altro"
 ELSE
  PRINT f;" :libs/";f$;".bmap"
 END IF
 READ f$
WEND
ch=0
WHILE ch<1 OR ch>f
 INPUT "La vostra scelta ";ch
WEND
IF ch<f THEN
 RESTORE
 FOR j=1 TO ch
  READ f$
 NEXT j
 nam$=":libs/"+f$+".bmap"
ELSE
 PRINT "Inserite il nome completo del file .bmap: ";
 LINE INPUT nam$
END IF
OPEN nam$ FOR INPUT AS #1
 bfile$=INPUT$(LOF(1),1)
CLOSE #1
sName=1
n$=CHR$(0)
WHILE sName<LEN(bfile$)
 eName=INSTR(MID$(bfile$,sName),n$)
 PRINT MID$(bfile$,sName,eName-1);TAB(25);
 ofset=CVI(MID$(bfile$,eName+sName,2))
 PRINT USING "#### ";ofset;
 sep$=" "
 parmPt=eName+sName+2
 parm=ASC(MID$(bfile$,parmPt))
 WHILE parm<>0
  parm=parm-1
  IF parm>7 THEN
   handle$="A"+CHR$(48+parm-8)
  ELSE
   handle$="D"+CHR$(48+parm)
  END IF
  PRINT sep$;handle$;
  sep$=","
  parmPt=parmPt+1
  parm=ASC(MID$(bfile$,parmPt))
 WEND
PRINT
sName=parmPt+1
WEND
```

Listato 5:

```
REM - questo programma dimostra come richiamare la
REM - routine EXECUTE dall'Amiga-BASIC

DIM x$(20)
PRINT "Generiamo una directory..."
DECLARE FUNCTION xOpen& LIBRARY
DECLARE FUNCTION Execute& LIBRARY
LIBRARY "dos.library"
z$=CHR$(0)
Hand1&=xOpen&(SADD("NIL:"+z$),1006)
x=Execute&(SADD("info> ram:temp"+z$), 0, Hand1& )
 OPEN "ram:temp" FOR INPUT AS #1
  PRINT 0;"<nessuno>"
  WHILE NOT EOF(1)
   LINE INPUT #1,x$
   IF MID$(x$,4,1)=":" THEN
    n=n+1
    x$(n)=LEFT$(x$,4)
    PRINT n;x$(n)
   END IF
  WEND
 CLOSE #1
KILL "ram:temp"
p=-1
WHILE p<0 OR p>n
 PRINT "La vostra scelta (da 0 a";n;")";
 INPUT p
WEND
```

continua a pag 79

Un secondo sguardo alla struttura dei file Amiga

di Betty Clay

Molti lettori di Transactor hanno familiarità con la configurazione dei dischetti formattati sui drive che hanno preceduto l'Amiga. Abbiamo studiato i libri di Dick Immers e Gerry Neufeld, abbiamo letto gli articoli di Mike Todd sul ICPUG, e abbiamo imparato cosa sono gli header gaps (aree di inizio file), i tail gaps (aree di fine file) e i GCR. Sappiamo che la directory si trova nella traccia 18 (o 39 sui drive 8050/8250), che il numero dei settori aumenta quando la testina si sposta verso l'esterno del disco e che i drive 4040, 2031, 1541 e 8050 scrivono solamente sul retro del disco. Molti di noi sanno come usare gli editor di tracce e settori su quei drive. Tale conoscenza ci può aiutare lavorando con l'Amiga, sebbene questi dischetti siano organizzati in maniera molto diversa.

Configurazione fisica

I dischetti Amiga vengono suddivisi in tracce e settori, ma in ogni lato del disco le tracce sono numerate da zero a settantanove. Ciascuna di queste tracce (160 in tutto) possiede undici settori. Due tracce, numerate nella stessa maniera e che si trovano su lati opposti del disco, formano un cilindro, per un totale di ventidue settori. Questi settori sono numerati sequenzialmente da zero a 1759, e il numero del settore viene registrato all'interno dello stesso. I cilindri vengono numerati dall'esterno verso l'interno, quindi il cilindro zero è il più esterno e il cilindro 79 è quello più vicino al centro. I settori iniziano al cilindro zero, settore zero, facciata zero (il lato superiore).

I drive hanno due testine che si muovono contemporaneamente, ma possono essere indirizzate individualmente. La testina numero zero si trova sulla facciata superiore del disco e la testina numero uno su quella inferiore. Le testine vengono spostate verso la traccia desiderata con il comando SEEK, e l'intera traccia viene letta con un comando CMD_READ oppure ETD_READ. Questi comandi, come pure altri, vengono usati dal trackdisk device per il controllo dei drive. Sono disponibili dei programmi per il controllo dei drive creati per programmatori C e in linguaggio assembler, ma non ho trovato il modo di creare i necessari file .bmap per controllarli da BASIC.

I dati vengono scritti in formato MFM (Modified Frequency Modulation o Modulazione di Frequenza Modificata) e vengono codificati o decodificati nel chip del blitter. È possibile predisporre il drive in maniera da scrivere in GCR, ma questo sistema legge e scrive a una velocità dimezzata rispetto al MFM. Qualcuno probabilmente troverà il modo di usare questo sistema in maniera da poter leggere i normali dischetti Commodore con l'Amiga, e non ci lamenteremo certo della velocità se questa possibilità diventasse reale.

Configurazione logica

Ognuno degli undici settori di una traccia ha un header (area di intestazione) di settore di 16 byte, più 16 byte riservati a un uso futuro, e 512 byte di "dati accessibili", per cui i nostri dischetti da 880K contengono anche 55K di dati per gli header, formando un totale di almeno 935K byte per disco (si pensa vi siano altri 28K). Non vi sono nè header gap nè tail gap, ma ogni traccia ha un gap costituito da una serie di byte null. Poiché ogni traccia ha lo stesso numero di settori, si potrebbe presumere che tale spazio diventi sempre più vasto man mano che si va verso l'esterno del disco.

L'area di header o label ha lo stesso schema per tutti i settori, ma i 512 byte di dati sono organizzati in maniera diversa a seconda del tipo di blocco. In un file di dati vi sono 24 byte di identificazione all'inizio del settore, seguti da 488 byte di dati effettivi. Un blocco che rappresenta una directory o l'header di un file ha 24 byte (sei long-word) utilizzati per il numero di settore, il tipo di file, il checksum ecc., all'inizio del blocco; 288 byte di hash-table o dati, e cinquanta long-word per il nome del file, il commento, la data, i puntatori in avanti e indietro ecc. alla fine del blocco.

Sui dischi Amiga alcune informazioni vengono codificate in byte da 8 bit, altre in word da 16 bit e altre ancora in long-word da 32 bit. Un "byte" equivale sempre a 8 bit, ma dobbiamo distinguere tra "word" e "long-word". Nel diagramma che segue tutte le "word" sono in effetti "long-word" formate da 32 bit.

Diagramma della configurazione dei blocchi

Per ogni tipo di settore:

```
Area Label:
2 byte 00
2 byte A1 (un sync byte MFM)
1 byte per il tipo di formattazione (FF su
    1.0)
1 byte per il numero della traccia
1 byte per il numero di settore
1 byte per l'offset -PIU' AVANTI SI PARLA
    ANCORA DI QUESTO ARGOMENTO!
16 byte per le informazioni per il recupero
    da parte del sistema operativo (attualmente
    non vengono utilizzati)
```

```
4 byte per il checksum dell'header
4 byte per il checksum dell'area dati
```

Per un blocco di dati, come un file sequenziale:

```
Area Dati:
1 word per il tipo di file
1 word per la chiave dell'header (settore
dove inizia il file)
1 word per il numero sequenziale di questo
blocco
1 word per il numero di byte di dati nel
blocco
1 word per il numero del blocco sucessivo nel
file
1 word per il checksum
fino a 488 byte di dati
```

O per un blocco riferito alla directory root

```
Area Dati:
1 word per il tipo di file
1 word per il numero di settore
1 word per la lunghezza del file
1 word per per le dimensioni dell'hash-table
1 word non utilizzata
72 word per l'hash-table
1 word per il flag di bit map
25 word per le pagine di bitmap (invece che
    BAM)
3 word per le ultime variazioni apportate
    alla data e all'ora
13 word per il nome del disco
3 word per la data e l'ora di crezione del
    disco
3 word per i puntatori in avanti e indietro
1 word per i file secondari
```

Il software è stato scritto tenendo conto di settori di diverse lunghezze che potrebbero trovarsi in versioni future. Invece di riferirsi a una word particolare si accede alle informazioni di identificazione che si trovano nelle ultime cinquanta word tramite "Block-size-n", dove "n" corrisponde al numero di long-word che precedono la fine del settore.

Come leggere il disco

Quando viene dato il comando SEEK, le testine si muovono contemporaneamente verso la traccia indicata. Nel momento in cui ricevono il comando READ, le testine leggono ininterrottamente per due rotazioni complete del disco, registrando il contenuto di un intero cilindro. Gli undici settori che si trovano su di una facciata del disco vengono registrati in un buffer e gli undici che si trovano sull'altra facciata vengono registrati in un secondo. Le testine non attendono un segnale di sincronizzazione, e non fanno nessuno sforzo per trovare un determinato settore. Non cercano neppure l'inizio di un settore! Leggono semplicemente i dati partendo da un punto della traccia in cui si trovano nel momento in cui viene dato il comando READ. I dati vengono decodificati dal blitter e i byte decifrati vengono messi nel track buffer nello stesso modo in cui escono dal disco. Questo tipo di lettura rende l'offset byte della label molto importante.

Il numero di offset

La prima volta che viene scritta una traccia su di un disco, verrà registrata con uno spazio di byte formati da null e poi gli undici settori in ordine numerico. Nel track buffer questa prima scrittura assume la forma seguente:

```
Settore N. 0 1 2 3 4 5 6 7 8 9 10 null
Offset N. 11 10 9 8 7 6 5 4 3 2 1 ....
```

e verrà codificato nel disco nello stesso ordine in cui è stato codificato nel buffer. Il numero di offset dice al data pointer quanti settori devono essere ancora letti o scritti prima di raggiungere lo spazio formato da null.

Tuttavia, quando questo settore viene riletto, è probabile che le testine non comincino a leggere dallo spazio e neppure dal settore 0. Supponete che all'inizio del READ il settore 7 si trovi sotto la testina. In questo caso il track buffer sarebbe:

```
Parte del settore 7 (dati confusi), e poi:
Settori N. 8 9 10 null 0 1 2 3 4 5 6 7 (altri
dati confusi)
Offset N. 3 2 1 .....11 10 9 8 7 6 5 4
```

Una volta che si trova nel track buffer, il pointer individuerà il primo sync mark e il blocco sposterà i dati nei settori 8, 9 e 10 in modo che il settore otto sia allineato con l'inizio del track buffer. Poi si sposterà oltre i null per individuare il synk mark seguente e spostare in su i restanti settori in modo che si uniscano ai primi, lasciando i null alla fine del buffer. Quando c'è bisogno di un settore, il software scandisce i settori del buffer e prende quelli necessari. Il disco non ha bisogno di essere letto di nuovo fino a quando non c'è bisogno di leggere una traccia diversa. Se i dati sono stati cambiati la traccia in uso viene riscritta sul disco prima che venga letta quella sucessiva.

Quando la nostra traccia ipotetica viene riscritta sul disco, i settori mantengono la numerazione originale, ma non la posizione originale. Quindi adesso gli offset di settore saranno modificati in modo da combaciare con il nuovo ordine dei settori:

```
Settori N. 8 9 10 0 1 2 3 4 5 6 7 null
Offset N. 11 10 9 8 7 6 5 4 3 2 1 .....
```

Ogni volta che la traccia viene riscritta, gli offset di settore verranno modificati, e il computer utilizza i numeri dell'offset per rintracciare i settori corretti. Questo metodo di scrivere i settori sulla traccia e la procedura di leggere ogni volta un'intera traccia evita la necessità di inserire dati qua e là ed evita header o tail gap, permettendo di registrare sul disco circa il venti per cento di informazioni in più. Leggendo un'intera traccia sequenzialmente l'accesso sul disco è più veloce ed efficiente.

Continua a pagina 28

Gioca sul serio!

D.O.C.

Se vuoi essere sempre super-informato e super-aggiornato, se vuoi divertirti e giocare, fallo sul serio.

Cerca sempre i prodotti Jackson Software D.O.C. perché solo Jackson ti garantisce serietà e giochi originalissimi, esperienza e novità, informazione e divertimento. E poi, diffida delle cattive imitazioni, delle cassette pirata e delle edizioni fantasma.

Se non vuoi brutte sorprese non scegliere a caso, non sprecare tempo e denaro. Chiedi sempre il marchio Jackson Software D.O.C. Scegli la sicurezza della qualità che è solo Jackson.

GRUPPO EDITORIALE JACKSON

AREA CONSUMER

Scegli il meglio: scegli Jackson.

Un cursore pilotato sull'Amiga?

...all'interno delle virgolette le cose funzionano in maniera leggermente diversa...

di Jim Butterfield

Inoltratevi nel CLI del vostro Amiga e scrivete il seguente comando:

```
ECHO HOT**DOG
```

Il computer, obbediente, ripeterà HOT**DOG per voi. Lo sapevate che il comando ECHO non ha bisogno delle virgolette, vero? La maggior parte delle volte vi trovate a usare le virgolette perché ciò che voi volete visualizzare contiene caratteri proibiti: nella maggior parte dei casi si tratta di spazi. È ragionevole.

Cerchiamo di essere più "formali" e scriviamo:

```
ECHO "HOT**DOG"
```

Stessa cosa, eccetto per il fatto che abbiamo messo la parola tra virgolette. Ma un momento...uno degli asterischi è scomparso! Che cosa sta succedendo?

Accade che quando vi trovate all'interno delle virgolette le cose funzionano in maniera leggermente diversa. Specialmente per gli asterischi: funzionano in maniera completamente diversa. Senza le virgolette gli asterischi sono come un qualsiasi altro carattere. Le virgolette creano un tipo di formato speciale. È come il "quote mode" (modo virgolette) o il "programmed cursor" (cursore pilotato) che avrete già visto sulle prime macchine della Commodore.

Trattamento speciale

Quando scrivete quasi ogni tipo di messaggio CLI all'interno delle virgolette, l'asterisco diventa un carattere simile all'escape. Probabilmente avrete visto qualcosa del genere nel C o in programmi come il PageSetter, eccetto per il fatto che un carattere ancora più usuale è la barra inversa (backslash). Questo significa: il prossimo carattere che seguirà non sarà un carattere normale, bensì un segnale speciale. Eccezione: quando scrivete il carattere due volte il secondo viene accettato come viene scritto, quindi i due asterischi diventano uno solo. Volendo potreste tradurre i due caratteri come segue: "Ecco un carattere speciale... Cioè un semplice asterisco".

L'asterisco è un residuo di un linguaggio di sistema di Amiga chiamato BCPL. Tale linguaggio fu un precursore del C, e scavando piuttosto a fondo ne troverete ancora dei resti all'interno dell'Amiga. Il BCPL originale aveva un numero elevato di codici speciali generati da un asterisco; nell'Amiga ne restano solo quattro:

```
**- genera un singolo asterisco;
*"- genera le virgolette
*n- genera una nuova riga (avanzamento di
      linea)
*e- genera un carattere escape ASCII.
```

Abbiamo già fatto delle prove con l'asterisco, ora potreste cercare di ottenere alcuni effetti interessanti con i due sucessivi scrivendo:

```
ECHO "PREMI*"RETURN*"QUANDO SEI PRONTO"
ECHO "LINEA 1*nLINEA 2"
```

Non sapevate di poter scrivere due linee con un solo comando ECHO, vero?

Ma la sequenza importante è *e, che genera il carattere di escape ESC. Probabilmente riconoscerete questo come CHR$(27) o, in numero esadecimale, 1B, ma se così non fosse non importa. Spesso ha effetti simili a quelli ottenuti usando il tasto ESC. Probabilmente saprete che premendo il tasto ESC seguito dal tasto "c" ripulite lo schermo del CLI. Vi aiuta anche a risolvere certi problemi fastidiosi, come ad esempio un particolare set di caratteri. Bene dunque: se *e tra virgolette equivale al tasto ESC, provate a fare:

```
ECHO "*ec"
```

Non vi siete sorpresi nel vedere lo schermo ripulito, vero?

Forse vi domanderete che importanza abbia tutto ciò. È più semplice premere due tasti, ESC e "c", che scrivere l'intera linea riportata qui sopra. Ma ECHO viene spesso usato nei batch file (che preferisco chiamare file EXECUTE, perché di solito li richiamate attraverso il comando EXECUTE). Quindi potete

fare in modo che una sequenza di comandi, non un programma, solo una sequenza di comandi, vi ripulisca lo schermo. Può anche fare altre cose per voi che spiegherò tra un momento.

Controsenso di un file

L'uso dell'asterisco, tuttavia, non è limitato al comando ECHO. Qualsiasi cosa voi mettiate tra virgolette utilizza le proprietà dell'asterisco o dell'escape. Quindi se volete chiamare un file CARO*PIPPO potete chiamarlo sia con quel nome... che "CARO**PIPPO". Ricordate: dovete raddoppiare l'asterisco solo se date il nome tra virgolette.

Potete anche inserire le virgolette nello stesso nome del file. Se volete chiamare un file "GATTO" (non GATTO ma "GATTO", in modo che le virgolette facciano parte del nome) potete farlo chiamandolo "*"GATTO*"". Sarà difficile che altre persone possano riferirsi al file con il suo nome. Sarà difficile che VOI riusciate a riferirvi al file con il suo nome. Non vorrete veramente fare questo, a meno che non desideriate dimostrare che può essere fatto.

Sono felice di comunicare che il DOS rifiuta nomi di file bizzarri, quelli cioè che cercherete di creare usando le sequenze utilizzate per il comando di "avanzamento di linea" o per l'escape. Dopo che ci avrete pensato, sarete lieti che le rifiuti.

Il CSI

Se generate il carattere ESC (e sappiamo come farlo, vero?) potete farlo seguire da un carattere di "aperta parentesi quadra" ("[") e tale combinazione si chiama CSI. CSI sta per Control Sequence Introducer, il che significa che dopo tale carattere seguirà qualcosa di speciale. Se capite il meccanismo, potete anche creare un CSI come carattere singolo, CHR$(155) o in numero esadecimale 9B.

Il CSI sembra una nuova forma di ESCAPE, ma ha un carattere abbastanza diverso (non intendevo fare un gioco di parole). Il comando ESC è seguito da un solo carattere attivo; il CSI può essere seguito da dati numerici (il che viene messo in rilievo) ma termina solo quando si raggiunge il carattere che definisce il comando. Il carattere comando di solito è alfabetico, ma fate attenzione: in questo caso c'è molta differenza tra i caratteri maiuscoli e quelli minuscoli.

Questa combinazione di un numero seguito da un comando è piuttosto simile a quella usata da ED, lo screen editor, per i comandi "estesi". Si tratta di quei comandi che si ottengono premendo il tasto ESC. Il loro formato, come nel CSI, è: prima scrivete il numero e poi la lettera di comando; il comando verrà eseguito per l'esatto numero di volte. Naturalmente il CSI ha i suoi comandi personali e non quelli utilizzati dall'ED.

Per esempio, il comando CSI per "avanzamento di linea" è "E" (notate che è maiuscolo). Quindi potete dare il comando:

```
ECHO "*e[6EBANANA"
```

Sarete soddisfatti nel vedere che il computer salta sei righe e poi scrive BANANA.

Eseguo solo un controllo per vedere se osservate la sintassi. Nell'esempio riportato qui sopra DOBBIAMO essere all'interno delle virgolette; l'asterisco indica che seguirà un carattere speciale; la e specifica che questa è una sequenza ESC; il carattere [dice: "ecco un comando speciale"; il 6 specifica il numero di volte che viene effettuata l'operazione susseguente; la E è il comando di avanzamento di linea, quindi eseguiamo sei avanzamenti di linea. E, come mi sembra abbia detto Freud, la BANANA è semplicemente una BANANA.

Se siete riusciti a capire questo, allora sappiate anche che F (sempre maiuscolo) è il comando utilizzato per un "avanzamento di linea" NEGATIVO. Provate a fare:

```
ECHO "*e[5FCIAO"
```

Osserverete qualche cosa che probabilmente consideravate impossibile; il cursore si sposta verso l'ALTO nello schermo del CLI! Facendo un po' di pratica vi troverete a scrivere sul vecchio testo che si trova sullo schermo, il che non è molto divertente... non avete ancora la possibilità di rielaborarlo come avrete probabilmente imparato dai primi computer della Commodore.

Non spiegherò dettagliatamente tutti i particolari comandi che possono essere richiamati con il CSI. Controllate nel manuale dell'AmigaDOS (Bantam Books); all'interno troverete un'appendice del Developer's Manual (Manuale per Programmatori). Scoprirete una quantità di comandi coi quali divertirvi e far fare delle cose strane allo schermo del CLI. Ve n'è uno in particolare che vorrei mostrarvi: il comando m (minuscolo) che imposta la rappresentazione grafica (Set Graphic Rendition).

Rappresentazione grafica

Il comando "CSI..numerico..m" cambia il metodo di visualizzazione sullo schermo del CLI. Il cambiamento, all'interno della finestra del CLI, è abbastanza stabile, per cui provo a modificare le cose già scritte al termine di ogni riga. Noterete due sequenze di *e[all'interno delle virgolette, una per produrre l'effetto desiderato e l'altra per tornare indietro di un certo numero di righe.

I codici che preferisco sono quelli della terza decina, vanno da 30 a 33 e modificano i colori di stampa (il 30 equivale al colore invisibile, quindi viene utilizzato di meno). Provate ora a fare:

```
ECHO "*e[32m CAVALLO NERO *e[31m"
ECHO "*e[33m BICICLETTE ROSSE *e[31m"
```

Se volete potete cambiare diversi colori in una sola riga.

I codici che si trovano nel gruppo della quarta decina cambiano il colore del fondo scrivendo:

```
ECHO "*e[41m INVISIBILE *e[40m"
ECHO "*e[42m SFONDO NERO *e[40m"
ECHO "*e[43m SFONDO ROSSO*e[40m"
```

Il risultato della prima linea non si vedrà molto bene poiché

verrà visualizzata in bianco su bianco. Osservate che qui è cambiato solo il colore del fondo.

Nella scala che va da 0 a 7 vi sono dei numeri molto interessanti. Provate a fare:

```
ECHO "*e[1m Neretto! *e[0m"
ECHO "*e[3m Corsivo! *e[0m"
ECHO "*e[4m Sottolineato! *e[0m"
ECHO "*e[7m Reverse! *e[0m"
```

Osservate che il codice 0 riporta il carattere al suo stato normale.

Un'ultima osservazione: potete usare diversi codici di questo tipo separati dal punto e virgola. Per finire con un'ultima combinazione provate a scrivere:

```
ECHO "*e[32mLeggete *e[3;4m Transactor
     *e[0;32;43m regolarmente!*e[0m"
```

Avrete notato che la sequenza *e[0m riporta tutto alla normalità. Quando l'abbiamo usata all'interno dell'ultima lunga linea per eliminare le caratteristiche del corsivo e del sottolineato, abbiamo dovuto ridefinire i colori che stavamo usando.

Nota a piè pagina per la stampante

Gli stessi codici CSI che funzionano con lo schermo, spesso funzionano anche con la stampante. Io uso frequentemente ECHO con il comando di redirezionamento () per scrivere un appunto veloce sulla stampante. Cioè:

```
ECHO > PRT: "Listato del 13 Marzo 1988"
```

Questo comando scriverà il messaggio corretto direttamente sulla stampante.

Ma è interessante osservare che molti codici per la stampante sono simili a quelli che abbiamo usato per lo schermo. La lista completa di tutti i codici di controllo per la stampante si trova sul manuale Amiga ROM Kernel ed è anche stata parzialmente riportata in numerosi altri testi. Un semplice esempio farà al caso nostro: se la vostra stampante supporta i caratteri corsivi, scrivete:

```
ECHO > PRT:"*E[3m Corsivo! *e[0m"
```

Lo stesso codice che ha funzionato sullo schermo funzionerà anche sulla stampante.

Conclusione

Potete senza dubbio ravvivare il vostro file Startup-Sequence con un po' di colore e altri effetti speciali. Potreste avere anche idee migliori su alcuni dei meravigliosi effetti che si trovano all'interno dell'Amiga.

(segue da pagina 24)

Un secondo sguardo alla struttura..

La codifica in MFM

Quando i dati vengono codificati in MFM, davanti a ogni bit viene messa una cifra aggiuntiva, per assicurarsi che non ci siano mai due 1 o due 0 in fila. Nell'MFM un bit "1" diventa "01". Un bit "0" sarà codificato come "00" se segue un bit "1", ma sarà codificato come "10" se segue un bit "0". Quindi, per ogni bit di dati che viene registrato, vengono scritti in realtà due bit. Il software di Amiga, utilizzando il blitter chip per codificare e decodificare, separa i bit dispari dai bit pari. Prima i bit dispari vengono codificati e scritti, poi i bit pari vengono spostati a sinistra di una posizione, codificati e scritti sul disco dietro ai bit dispari. Quando i dati vengono riletti, il blitter chip elimina gli zeri e gli uni che si trovano davanti e che sono stati aggiunti per la codifica, lasciando degli "spazi" tra i bit dispari in modo che vi si possano inserire i bit pari.

Il procedimento di codifica per le etichette di settore è interessante. I primi quattro byte della label vengono codificati come byte separati, prima i bit dispari e poi i bit pari di ogni byte. Poi i sucessivi quattro (la formattazione, il numero di traccia, il numero di settore e l'offset) vengono codificati come una long-word. I 16 byte riguardanti le informazioni sul recupero del sistema operativo vengono codificati come blocco di 16 byte. Il checksum dell'header e il checksum dell'area-dati vengono codificati ciascuno come una long-word. I 512 byte del blocco-dati vengono codificati come un singolo blocco di dati, prima i bit dispari di tutti i 512 byte e poi i bit pari.

Un po' di matematica

I dischi sono regolati per ruotare a 300 giri al minuto. Con la codifica GCR l'Amiga scrive a una velocità di quattro microsecondi per bit. In MFM richiede solo due microsecondi per bit. A 300 rivoluzioni al minuto si avrebbero cinque giri al secondo, oppure duecento millisecondi a rivoluzione. Due microsecondi per bit permettono di scrivere 100.000 bit a rivoluzione. Ci vogliono due bit MFM per ogni reale bit di dati, sebbene sia permesso un massimo di 50.000 bit di dati per traccia. Abbiamo 160 tracce, cioè 8.000.000 di bit di dati possibili, oppure 1.000.000 di byte per disco. Abbiamo dato un significato a 957.440 byte (935 * 1024), escludendo gli spazi dei null. Non è un'efficienza straordinaria?

E un mistero

C'è un mistero riguardo alle label di settore. Il ROM KERNEL MANUAL dice che c'è una sezione di 16 byte di dati descrittivi per ogni settore, per un totale di 27,5K byte. Il drive, dice il manuale, non interpreta queste sezioni a meno che il programmatore non gli abbia dato istruzioni per fare ciò. Non si può trattare dell'area label descritta qui sopra, poiché le informazioni in tale area devono essere interpretate per le normali operazioni del drive. Si riferisce dunque ai 16 byte che il RKM dichiara inutilizzati, ma che devono essere usati per le informazioni di recupero del sistema operativo? Oppure esiste un'altra area label descrittiva? VOI avete la risposta a questo mistero?

Qualche esempio di programmazione della shell

Testi e coreografia

di John Faichney

John Faichney lavora presso il The Arts Television Centre a Toronto. In questo articolo, John ci spiegherà come sia possibile estendere la funzionalità degli strumenti forniti con il sistema operativo dell'Amiga.

Questo articolo mette in evidenza esempi di programmi di shell, file formati da istruzioni che, in seguito alla loro esecuzione, lanciano dei normali comandi di AmigaDOS. Prima di tutto bisogna fare una puntualizzazione: nessuno dei programmi contenuti in questi articoli contiene del codice *oggetto*, ma solo alcune sequenze di controllo da usare alla console. Inoltre, nessuno di essi contiene dei comandi non standard di AmigaDOS v1.3, anche in quei casi in cui il loro uso avrebbe reso più facile la stesura dei programmi.

Molti esempi sono rivolti a tipici problemi di gestione dei file e dei dischi e possono essere di interesse pratico per gli utenti il cui spazio di indirizzamento e/o memoria del file-system siano delle risorse scarse; per quelli a cui non interessa il *codice oggetto* legato alla gestione dei file e dei dischi (DirUtil, DiskMan, e simili). Altri esempi, sebbene perfettamente funzionanti, avranno come interesse primario la dimostrazione delle tecniche di programmazione coinvolte. Altri ancora non saranno altro che esempi di programmazione della shell di un certo peso.

1. Un importante esempio di programmazione della shell

Inserite la seguente linea dopo il prompt della shell (1.RAM DISK:>):

```
1.RAM DISK:> Join * as Prompt.test
```

Quindi digitate:

```
Prompt "ESC[31m%n.%s>ESC[32m"
```

Notate che il tasto ESC non viene stampato sullo schermo. Battete la sequenza di tasti CTRL-\ per inviare un end-of-file al comando Join. Inserite ora:

```
1.RAM DISK:> Execute Prompt.test
```

il cui effetto sarà quello di colorare la finestra della shell e il prompt con il colore 1 del Workbench, e l'I/O della linea di comando con il colore 2.

2. Come creare e stampare dei file di testo con Join ed Echo

Come è stato dimostrato nel paragrafo precedente, passando l'argomento *finestra corrente* (un asterisco) a Join come suo parametro di *file di input*, si possono creare dei file di testo come Prompt.test direttamente dalla shell. Specificando la finestra corrente come il parametro AS (file di destinazione) di Join si crea un comando simile al Type:

```
1.RAM DISK:> Join Prompt.test as *
```

Sebbene Join non richieda più di un argomento come suo parametro di *file di input*, può essere utilizzato ugualmente per concatenare più occorrenze della finestra corrente:

```
1.RAM DISK:> Join * * * * * * * * * * as foo
```

anche se difficilmente questo è il modo più intuibile per creare un file di testo di 2550 (dieci x 255 caratteri per CTRL-\) byte. Consideriamo, però, il seguente esempio:

```
1.RAM DISK:> Join * Prompt.test as S:StandOut
```

inserite ora:

```
;Un importante esempio di programmazione
        della shell
```

e digitate CTRL-\. Usando Type, questa volta, per visualizzare il file:

```
1.RAM DISK:> Type S:StandOut number
```

otterrete:

```
1 ;An outstanding example of shell programming
2 Prompt "%n.%s>"
```

Come dimostrano questi esempi, i comandi che normalmente dirigono i loro flussi di I/O da/al file-system possono facilmente essere redirezionati verso altri tipi di dispositivi.

"E allora", potreste dire, "sicuramente la programmazione della shell ha associate potenzialità che vanno ben oltre i pochi trucchi usati per creare dei semplici file di testo."

Bisogna tener presente però, che questi pochi trucchi per la creazione di semplici file saranno disponibili ogni volta che ci sarà da provare una combinazione di condizioni, parametri ecc. senza aver bisogno delle prestazioni di un intero editor. Anche se non è molto importante il modo in cui vengono scritti i file di testo, è comunque opportuno fare alcune osservazioni su come i testi saranno presentati:

1) Tutti i testi avranno i numeri di linea, e, dove necessario, i fine linea; queste caratteristiche non sono peculiari dei file di comandi.

2) Sia il delimitatore di linea dei comandi ';'che la direttiva '.' (punto più spazio) sono usate per distinguere i commenti frammisti al codice.

Notate comunque che, dalla versione 1.2 in poi, non tutti i comandi gradiscono il carattere ';' come delimitatore della linea dei comandi (vedere Else ed Endif e l'esecuzione di comandi come singolo processo in background e concatenati con il carattere '+'). Inoltre, certi comandi o certe direttive (vedi punto più spazio) non vogliono essere preceduti e/o seguiti da caratteri di tabulazione. Questi cavilli sono così banali che, nonostante siano dei malfunzionamenti, non dovrebbero comunque creare grossi problemi; in ogni caso chiunque stia sviluppando dei file di comandi sotto la versione 1.2, e voglia aggiungere dei commenti e/o dei caratteri bianchi (diversi dal carattere spazio) alle linee dei comandi, è tenuto a controllare possibili effetti collaterali o, in alternativa, a tenerli su linee separate.

3) I comandi possono essere indentati usando il carattere spazio.

Concludendo, gli utenti che usano dei word-processor per sviluppare dei file di comandi dovrebbero controllare, per esempio, che l'output del file non tolga le direttive punto.

Il passare l'argomento *finestra corrente* come parametro di un flusso di input ha un effetto collaterale non desiderato: la finestra corrente non è più disponibile come interfaccia per la linea di comando. Eseguendo il comando con Run (cioè come processo in background) non funzionerebbe ugualmente (nemmeno, se ci pensate, quando vorremmo che lo facesse). Usare il comando Run funziona se si riceve il flusso di input da una finestra associta alla console.

```
1.RAM DISK:> Run Join
 NewCon:5/80/629/45/my_window as foo
```

Il lettore potrebbe scegliere di posizionare o di dimensionare la finestra in un modo diverso.

Sebbene questo metodo funzioni, la sua lunghezza lo rende molto meno utile del comando che sostituisce. La soluzione più efficente è quella di inserire il comando all'interno di un file di comandi.

```
1 . GetText
2 .key AS/a
3 Join "NewCon:5/80/629/45/Inserire il testo
(al massimo 255 caratteri) ; [CTRL-\] per
terminare. " as "<AS>"
```

GetText mette in evidenza (linea 2) una dichiarazione di parametri o 'template' (maschera) per l'argomento della linea di comando che verrà sostituito nella linea 3; in questo caso il parametro AS di Join. Come per i normali comandi di AmigaDOS, il suffisso "/a" indica che si desidera che questo file di comandi fallisca nel caso in cui venga eseguito senza un argomento. Da notare anche che il parametro AS nella linea 3 è racchiuso tra virgolette, in modo tale che l'utente abbia la possibilità di inserire un nome di file contenente degli spazi e dei caratteri speciali.

La linea 3 passa a NewCon: anche il titolo della finestra. Notate che l'uso dei caratteri spazio all'interno di questa stringa forzi che l'intera specifica della finestra venga racchiusa tra virgolette.

Si può ora mandare in esecuzione il file digitando

```
1.RAM DISK:> Run Execute GetText foo
```

e la nostra finestra associata alla shell rimarrà attiva in attesa di processare altre linee di comando.

Si può creare un semplice file testo anche utilizzando il comando Echo:

```
1.RAM DISK:> Echo > foo "Testo di prova
    generato con ECHO"
```

non c'è neanche la necessità di usare la sequenza CTRL-\ per inviare un end-of-file. Comunque, essendo il comando Echo orientato più alle stringhe che ai file vengono imposti dei limiti al tipo di I/O da esso supportato. Infatti, può sembrare che Echo non riesca ad esprimere la redirezione dell'input; l'uso di

```
1.RAM DISK:> Echo < foo ?
```

da come risultato il messaggio

```
:Invalid argument to Echo
```

Il problema risiede nel fatto che la stringa "Testo di prova generato con ECHO" perde le sue virgolette all'interno del file. Si ha bisogno di:

```
1.RAM DISK:> Echo > bar
    "*"Questo è un nuovo testo di prova
      generato con ECHO.*""
```

nel quale le virgolette precedute dall'asterisco vengono realmente scritte nel file bar, mantenendo l'integrità della stringa. Ma sicuramente questo è un modo molto scomodo per generare un file di testo. Difatti, una delle cose più belle nel passare dei testi a dei comandi che orientano i loro flussi di input a dei file è la *resistenza* di questi comandi verso dei metacaratteri come newline e virgolette non chiuse: sequenze di caratteri che Echo ha molte difficoltà a capire ma che capitano abbastanza spesso (se non altro per errori di battitura) quando l'input arriva dalla tastiera.

Da notare, tra parentesi, che si possono anche usare le 'pipe'- come contenitori *volanti* per il testo; per esempio:

```
1.RAM DISK:> Echo > Pipe:foo "
     Echo *"Questo foo è diverso dal vecchio
     foo e anche da bar- *
     **"Ciao a tutti"*** *""
```

Inoltre, si possono eseguire questi programmi che usano le pipe propio come si farebbe con quelli che usano i file. Ricordate, comunque, che una Pipe: dell'Amiga è un dispositivo paragonabile ad un flip-flop; una volta riempito non può essere riempito nuovamente fino a che non viene svuotato; una volta svuotato deve essere comunque riempito prima di essere risvuotato.

Diamo ora alcune indicazioni per quello che riguarda l'argomento '?'.

2a. L'argomento '?'

L'esperienza ci insegna che la shell interpreta la linee di comandi più o meno secondo la seguente procedura:

1) acquisisce la linea di comando;
2) trova il file indicato dalla prima stringa della linea di comando e controlla che sia eseguibile;
3) carica il comando dalla directory specificata o da quella implicita;
4) se è presente l'argomento '?':

- invia il template (maschera) del comando al flusso di output specificato con il comando (se non viene specificato un flusso di output, allora per default viene assunta la finestra corrente)
- appende all'argomento(i) fornito con il comando il flusso di input fissato nel comando (se non viene specificato nessun flusso di input, allora per default viene assunta la finestra corrente) e sospende l'esecuzione fino a che non viene chiuso il flusso di input;
- quando il flusso di input viene chiuso, esegue il comando.

Molti dei file di comandi presentati in questo articolo usano l'argomento '?' per forzare la redirezione dell'input, a volte da una finestra, altre dal file system. Alcuni utilizzano la sospensione dell'esecuzione, forzata da '?', quando il flusso di input è la console. Altri usano specificamente il fatto che il comando sospeso (o file di comandi) non è caricato nello spazio di indirizzamento prima di essere sospeso. Altri ancora combinano la redirezione dell'input con la stampa di una stringa secondaria di prompt con funzione *descrittiva.* Altri infine usano la redirezione della maschera del comando verso il file system.

Molti addetti ai lavori hanno interpretato l'integrazione dell'argomento '?' nell'AmigaDOS come un sistema di help in linea molto scarno. Un tipo di interpretazione di questo genere sembrerebbe trascurare una parte fondamentale delle sue funzionalità

2b. Ancora sulla creazione dei file di testo con Join ed Echo

GetText è limitato dal fatto che si possono inserire al massimo 255 caratteri per volta, senza premere RETURN. Si può comunque scavalcare questa limitazione e descrivere altre tecniche di programmazione al riguardo. Un modo è quello di ricevere l'input da più di una finestra; qualcosa del tipo:

```
1 . MoreText
2 .key AS/a
3 Join "NewCon:5/11/629/45/Inserire il testo
    (al massimo 255 caratteri) ; [CTRL-\] per
    terminare." "NewCon:5/45/629/45/Inserire
    altro testo (al massimo 255 caratteri) ;
    [CTRL-\] per terminare." as "<AS>"
```

Ma è difficile vederne un ulteriore sviluppo. Quello di cui si ha bisogno è di un file di comandi che accetti un input, indefinitivamente, fino a che l'utente non decida di fermarsi. Il file seguente, Texter, serve a questo.

```
1 . Texter
2 .key TO/a,SKIP/s,FILE.0/k,FILE.1/k
3 .bra {
4 .ket }
5 {SKIP}
6 Run > Nil: Echo > Nil: <
    "Con:5/59/629/25/Se non c'è più testo per
    {TO} , battere [RETURN] in questa
    finestra."?+
7 Echo > :t/texter.d
8 Lab
9 If exists :t/texter.d
10    Join {FILE.0$Nil:} as "{TO}"
11    Delete > Nil: :t/texter.?
12 Else
13    Join (FILE.0} "NewCon:5/80/629/45/Inserire
    il testo (al massimo 255 caratteri) ;
    [CTRL-\] per finire." as {FILE.1$:t/texter.1}
14  Execute Texter "{TO}" skip file.0
    (FILE.1$:t/texter.1} file.1
    {FILE.0$:t/texter.0}
15 EndIf
```

La strategia di Texter è a due fasi. Per prima cosa, Texter sfrutta la ricorsione per creare una sequenza indefinita di finestre per la console entro le quali possa essere inserito il testo; non appena ogni sezione del testo viene chiusa, è aggiunta al file relativo alla sezione precedente in modo da creare un nuovo file, e viene aperta una nuova finestra. In secondo luogo, Texter mette in moto e poi sospende l'esecuzione di un processo in background il quale, in base a una riattivazione da parte dell'u-

tente, termina il loop ricorsivo di Texter.

Come ci si doveva aspettare, il processo in background è fatto partire per primo, nelle linee 6 e 7. Consideriamo per adesso che l'utente non abbia ancora risposto alla finestra del processo in background (ci ritorneremo tra un attimo), il controllo del processo in foreground passa alla linea 13.

La prima volta che Texter viene eseguito, non viene passato nessun parametro a FILE.0, e quindi quest'ultimo non appare sulla linea 13; è semplicemente scaricato. L'utente ha a che fare con la finestra della console (per ora familiare), il cui input deve terminare con CTRL-\ . Invece, (sempre durante la prima esecuzione) non viene passato nessun argomento a FILE.1, e quindi viene scaricato, ma con la differenza che il file temporaneo

```
:t/texter.1
```

è inserito al suo posto (notate il flag parametro di default, '$'), in modo tale che l'input dell'utente venga scritto in questo file.

A questo punto Texter usa la ricorsione (linea 14), passandosi quello che l'utente aveva inserito come parametro TO. Vengono anche passati parametri generati dalla linea 14 stessa, incluso SKIP. Texter sfrutta l'occorrenza di SKIP come modo per rilevare se egli stesso sia stato mandato in esecuzione dalla linea di comando o a causa della ricorsione; cioè, come modo per assicurare che il processo in background venga eseguito una sola volta, a prescindere da quante volte Texter usi la ricorsione.

Ritornando alla linea 14; siccome questa è ancora l'esecuzione iniziale di Texter, gli argomenti che accompagnano le keyword FILE.0 e FILE.1 consistono nei parametri di default,

```
:t/texter.1
:t/texter.0
```

rispettivamente; gli argomenti per FILE.0 e FILE.1 appariranno nella linea 14 non prima della prossima esecuzione. Notate, comunque, l'associazione della keyword "file.0" con l'argomento {FILE.1$Pipe:texter.1} e l' associazione della keyword "file.1" con l'argomento {FILE.0$Pipe:texter.0}; cioè, l'associazione delle keyword e dei file temporanei è rovesciata ogni volta che Texter usa la ricorsione. Ed è proprio questa alternanza delle associazioni di keyword che permette a Texter di (nella linea 13) concatenare indefinitivamente l'input dell'utente in una delle pipe.

Sembra troppo complicato ? Proviamo un'altra volta:

Supponiamo di chiamare la prima/terza/quinta ecc. iterazione di Texter, come iterazioni dispari, e la seconda/quarta/ecc. come quelle pari. Ora, per ogni iterazione dispari, la linea 13 diventa:

```
13 Join :t/texter.1 "NewCon:...per finire."
     as :t/texter.0
```

Prende il contenuto attuale di :t/texter.1, riceve del testo dalla window, e pone il risultato della loro concatenazione in :t/texter.0. In modo analogo, per ogni iterazione pari di Texter la linea 13 diventa:

```
13 Join :t/texter.0 "NewCon:...per finire."
     as :t/texter.1
```

così viene aggiornato :t/texter.1. Sebbene l'assegnazione di default per gli argomenti ai parametri di FILE.0 e FILE.1 venga fissata durante l'esecuzione iniziale di Texter, in seguito il passaggio degli argomenti a FILE.0 e FILE.1 viene gestito in modo dinamico, come conseguenza dell'alternanza di associazioni generata dalla linea 14.

Questo processo di aggiornamento alternato dei file temporanei di Texter continua indefinitivamente, anzi non c'è modo di uscire dall'iterazione dal processo nel quale è stato chiamato Texter. Invece, Texter controlla (linea 9) l'esistenza di un file dummy:

```
:t/texter.d
```

il quale può essere creato da qualche altro processo concorrente o, in modo più specifico, dal processo in background lanciato da Texter stesso.

Ricordate che la linea 6 rimane sospesa fino a che l'utente non rende attiva la finestra di Echo e digita in essa RETURN (dato che non si inserisce del testo, non si fa uso del gestore di console avanzato). Viene ripresa l'esecuzione del processo in background, generando la creazione del file dummy (linea 7). Il controllo, all'interno del processo in foreground, passa quindi alla linea 10, dove l'aggiornamento più recente dei file temporanei di Texter viene copiato nel file TO specificato dall'utente. Texter cancella i file temporanei (linea 11) e termina.

Texter presenta, rispetto a GetText, un notevole incremento della complessità, che è dovuto in parte ai dettagli presenti. Per esempio, tre su quattro dei parametri di Texter, "SKIP", "FILE.0" e "FILE.1" vengono usati per usi personali. Da notare che SKIP è un *interruttore*; cioè, se "skip" compare come argomento della linea di comandi, allora la keyword "{SKIP}" della linea 5 è *riempita*; altrimenti l'occorrenza della keyword, cioè la linea 5 stessa, è semplicemente lasciata in bianco.

Le linee 3 e 4 definiscono i caratteri parentesi angolari. Come si è visto prima, la shell usa i caratteri '<'e '>'per attivare la redirezione dell'input/output. Questi sono gli stessi caratteri che vengono usati per distinguere le presenze di keyword nelle linee dei file di comandi. Senza la definizione delle parentesi angolari, la shell non permetterebbe la redirezione dell'input (anche se permetterebbe quella dell'output.) La linea 6 utilizza la redirezione dell'input; da qui la necessità delle linee 3 e 4.

Molti dei listati presentati in questo articolo redirigono l'output della console verso il dispositivo NIL: ; per esempio la linea 6 di Texter provvede alla redirezione sia per Run che per Echo. In questi e in altri casi, la visualizzazione da parte della console di prompt secondari o di messaggi di errore non serve per nessuno scopo utile alla gestione dell'interazione dell'utente

con il file di comandi. D'altra parte, l'uso di NIL: sulla linea 10 è puramente a scopo difesivo; l'utente potrebbe rispondere alla linea 10 in modo così veloce da creare :t/texter.d prima che Texter abbia avuto la possibilità di innescare la ricorsione, bisogna quindi fornire un argomento di default per l'input che prenda il posto di FILE.0.

Nonostante che Texter abbia una qualche eleganza nel risolvere il problema di concatenazione di un testo con lunghezza indefinita, il suo metodo di terminare la ricorsione è scarno (come si vedrà, potremo dire ciò di molti altri file di comandi presentati). Inoltre, si potrebbe cercare qualche altro metodo in sostituzione al fatto che l'utente debba inviare un end-of-file in ogni sezione del testo; la combinazione di tasti CTRL-\ è scomoda e poco intuitiva. Anche se l'uso di un end-of-file assicura il trattamento non traumatico delle sequenze di controllo e permetta sia dei newline completi, sia la continuazione oltre i limiti di una sezione di una stessa linea di testo, si potrebbe voler rinunciare a qualcuna di queste funzionalità in favore di una facilità d'uso associata alla chiusura di una sezione di testo con un semplice RETURN.

Infine, l'uso fatto da parte di Texter dei file temporanei è tale che si potrebbe preferire di rimpiazzarli con delle pipe.

Di conseguenza, ecco Liner.

```
1 . Liner
2 .key TO/a,SKIP/s,FILE.0/k,FILE.1/k
3 .bra {
4 .ket }
5 .FailAt 21
6 {SKIP}
7 Run > Nil: Echo > Pipe:liner.d <
  "Con:5/69/629/25/Se non c'è altro testo per
  {TO}, battere [RETURN] in questa finestra."?+
8 Join Pipe:liner.d as Nil:
9 Lab
10 Join > Nil: Nil: as Pipe:liner.d
11 If not fail
12   Join {FILE.0$:Nil:} as "{TO}"
13   Join Pipe:liner.d as Nil:
14 Else
15   Execute GetArgs Pipe:liner.n prompt
     "Inserire il testo per {TO} (al massimo 15
     stringhe di 255 caratteri.)"
16Join {FILE.0} Pipe:liner.n as
  {FILE.1$Pipe:liner.1}
17Execute Liner "{TO}" skip file.0
  {FILE.1$Pipe:liner.1} file.1
  {FILE.0$Pipe:liner.0}
18 EndIf
```

Liner è costruito sullo stile di Texter, con alcune piccole ma importanti differenze. Come si può notare dal listato, non ci sono più dei file temporanei; le pipe, invece, vengono usate sia come buffer per il testo che come marcatore per il processo in background. Il primo dei due listati si documenta abbastanza da solo; quest'ultimo introduce delle nuove complicazioni sulla programmazione della shell.

Nella linea 7, Echo apre una pipe e rimane sospeso aspettando che l'utente digiti un RETURN nella finestra di Echo, al che (linea 8) la pipe viene svuotata. Come si faceva notare precedentemente, è una caratteristica della pipe il fatto che, una volta aperta, nessun altro processo la può riempire fino a che non viene svuotata. La linea 10 perciò fallisce fino a che l'utente non risponde, e Liner è in grado di scoprire (linea 11) lo stato del processo in background.

Esiste però un'altra differenza fondamentale. Nello stesso punto dove Texter aveva ricevuto del nuovo input, per Join dalla finestra della console, da parte dell'utente, Liner riceve l'input da una pipe.

```
Pipe:liner.n
```

Ma la pipe da dove prende il suo input ?

Nella linea 13, Liner richiama un altro file di comandi, GetArgs, per ricevere l'input dall'utente (notate che Liner fornisce a GetArgs il nome della sua pipe, e una stringa di prompt). GetArgs è eseguito nello stesso processo di Liner, cioè, una volta che Liner ha chiamato GetArgs, non ricomincia finché GetArgs non è terminato, chiaramente, GetArgs è il punto focale del programma.

GetArgs è composto da queste istruzioni piuttosto criptiche:

```
1 . GetArgs
2 .key TO/a,PROMPT/k
3 .bra {
4 .ket }
5 FailAt 21
6  Execute > Nil: <
  "NewCon:5/94/629/25/{PROMPT}" PutArgs"{TO}"?
7 If fail
8   Execute GetArgs"{TO} "Dato non accettato.
    {PROMPT}"
9 Endif
```

Ma neppure GetArgs è il termine ultimo del processo; GetArgs richiama a sua volta (linea 6) un altro file, PutArgs. Infatti, nonostante sia GetArgs che apre una finestra per l'input, qualsiasi cosa venga inserita nella finestra viene passata a PutArgs come argomento. Mentre i file di comandi precedenti utilizzavano la redirezione dell'input verso dei comandi di AmigaDOS, GetArgs usa la redirezione verso un altro file di comandi. Osservate anche come la linea 6 contenga pure gli argomenti che erano stati passati a GetArgs: l'argomento "Pipe:liner.n" di Liner (Liner, linea 15) è passato semplicemente a PutArgs come uno dei suoi argomenti, mentre la stringa di prompt di Liner è presente in NewCon:barra di titolo di GetArgs.

Il perché si debba usare PutArgs in un modo così decisamente complicato diventa più chiaro dando un'occhiata a PutArgs stesso.

```
1 . PutArgs
2 ,key _0/a,_1,_2,_3,_4,_5,_6,_7,_8,_9,
       _A,_B,_C,_D,_E,_F
```

```
3 Echo > "<_0>" "<_1> <_2> <_3> <_4> <_5>
    <_6> <_7> <_8> <_9> <_A> <_B> <_C> <_D>
    <_E> <_F>"
```

Quando GetArgs chiama PutArgs, "Pipe:liner.n" di Liner diventa il primo degli argomenti passati a PutArgs; cioè viene associato con il parametro _0, file di destinazione di PutArgs. PutArgs scrive fino a quindici stringhe su _0; le rimanenti arrivano dalla finestra di GetArgs. Perché propio quindici? A suo tempo sembrava una buona idea. Notate che se l'utente inserisce una stringa identica a qualsiasi delle keyword presenti nella dichiarazione di PutArgs, la stringa sarà assegnata a quella keyword e l'ordine delle stringhe verrà alterato. Questo è un problema per il quale non esiste alcuna soluzione; tutto quello che si può fare è di usare una keyword abbastanza inusuale come "_0", che si suppone non capiti quasi mai.

Quindi, niente ci impedisce di eseguire PutArgs direttamente dalla linea di comando:

```
1.RAM DISK:> Execute PutArgs foo string_1
     string_2 ... string_15
```

anche se Echo, da solo, avrebbe fatto la stessa cosa, come si è visto precedentemente. Comunque, dal momento che Echo richiede che l'utente inserisca delle virgolette, PutArgs permette la richiesta di input dell'utente come una serie di stringhe separate. Inoltre, il fatto che GetArgs utilizzi la ricorsione (GetArgs, linee 6, 7, e 8) potrebbe far si che PutArgs fallisca a causa di un input dell'utente non corretto (metacaratteri, troppe stringhe, ecc.). In questo caso, GetArgs tenta di assistere l'utente inviandogli un prompt esplicativo (GetArgs, linea 8); questo chiarimento, per quanto riguarda la ricorsione, è inserito nell'argomento PROMPT di GetArgs.

Perché in Texter non è stata inserita nessuna istruzione FailAt, nonostante richieda l'input attraverso l'uso di Echo ? Per il fatto che il *livello di insuccesso* associato al comando Echo non è così alto da impedire l'esecuzione dei comandi successivi. Potrebbe sembrare che un input difettoso verso PutArgs faccia crollare anche GetArgs come conseguenza indiretta della caduta di Echo in PutArgs, ma è lo stesso Execute di GetArgs che rifiuta l'input non valido, e impedisce che Echo sappia che è stato ricevuto un input.

Ritornando a Liner, si vede che chiama GetArgs (Liner, linea 13), passandogli il nome della pipe alla quale bisogna inviare l'input di GetArgs (cioè, Pipe:liner.n), e una stringa di prompt. GetArgs semplicemente passa il nome della pipe a PutArgs e crea una finestra NewCon: dalla quale PutArgs riceverà il resto del suo input (GetArgs, linea 6). PutArgs accetta l'input, crea il file e termina, restituendo il controllo a GetArgs. Poichè GetArgs non ha altro da fare, il controllo ritorna a Liner e il contenuto della pipe appena riempita è svuotato nella corrente pipe FILE.0 di Liner (Pipe:liner.0 o Pipe:liner.1, in funzione di quale dei due comandi ricorsivi aveva chiamato GetArgs). A causa di questa strategia di doppia indirezione, Liner gestisce inserzioni di testo una riga alla volta; e siccome tutto il testo viene trattato come input di stringhe, non è necessario l'end-of-file (CTRL-\).

3. Una digressione sulla gestione dei file temporanei

Texter e Liner si basano pesantemente sull'uso di file temporanei e sulle pipe. In qualche caso, questo tipo di risorse viene usato per *emulare* flag e variabili. Forse potrebbe sembrare come se ciò andasse un po' oltre il loro vero utilizzo, ma cerchiamo di capire che cosa sono in grado di fare questi strumenti, anche se utilizzati a volte in modo non lineare.

Un problema generale legato all'utilizzo di file temporanei e di pipe è la loro vulnerabilità all'alterazione in ambienti di multiprogrammazione. Si consideri, per esempio, lo stesso comando Execute: chiaramente non funziona molto bene quando delle versioni temporanee di file di comandi (con i parametri già sostituiti) vengono *sporcate* da un utilizzo dello stesso nome come risultato, per esempio, di una esecuzione concorrente dello stesso file. La stessa cosa potrebbe essere detta a riguardo dei file/pipe usati dai file di comandi, prescindendo dai loro scopi, e, infine, la completa fattibilità di file di comandi che usano dei file/pipe in un ambiente multitasking risiede nella possibilità di assicurare, in qualche modo, l'unicità del nome del file. Se questo obiettivo non può essere raggiunto, si sarà costretti a eseguire un file di comandi alla volta, sullo stile MS-DOS. Ma sicuramente si può fare meglio.

Una futura puntata di questa serie di articoli sulla programmazione della shell potrebbe essere completamente dedicata alla realizzazione di un programma basato sulla shell per la soluzione di questo problema.

4. Come gestire le variazioni delle condizioni di startup

Sebbene dei programmi di utilità rivolti all'inserimento di testo come Texter e Liner possono essere usati in molti contesti diversi, il loro utilizzo della concorrenza di processi porta a un degrado delle prestazioni se ciascun elemento del loro insieme di comandi deve essere caricato da floppy disk. Se rendiamo residente il loro insieme di comandi durante il boot, le prestazioni sono accettabili. Comunque, si potrebbe volere che l'insieme dei comandi non sia sempre residente; è meglio salvare la Startup-Sequence originale come, diciamo, "Startup-Sequence.old" e creare una nuova Startup-Sequence che contenga le chiamate al comando Resident. In seguito, se si vuole ritornare alla vecchia Startup-Sequence, si inventerà un nome per chiamare la Startup-Sequence corrente (la chiameremo "DoResident-Sequence") e si invertiranno i nomi:

```
1.RAM DISK:> Rename S:Startup-Sequence as
    S:DoResident-Sequence
1.RAM DISK:> Rename S:Startup-Sequence.old as
    S:Startup-Sequence
```

Come soluzione più generale, si potrebbe dedicare una directory in :s (chiamiamola "seqs") per memorizzare le startup-Sequence sostitutive. Si potrebbe tenere un originale di ciascun file in :s/seqs e (per effettuare il cambio di startup-Sequence) copiare dall'originale in :s/Startup-Sequence. Così si tengono nascoste le sequenze non usate correntemente e si prevengono le loro possibili cancellazioni o esecuzioni.

Sfortunatamente, con questa soluzione il nostro disco di boot

contiene due copie del file di boot, una copia originale, e quella su cui si sta lavorando attualmente. Inoltre, è sicuramente inefficente riscrivere l'informazione; si deve poter cambiare nome sia all'originale sia alla copia di lavoro senza perdere traccia del loro nome originale. Si potrebbe mettere il nome in un file, ma la gestione di troppe directory e file potrebbe risultare troppo confusionaria. Quello di cui si ha bisogno è di un programma di utilità che ricordi il nome del file originale e che effettui lo scambio di Startup-Sequence. Il file di comandi seguente, ChangeSeq, fa esattamente queste cose.

4a. ChangeSeq, un programma di utilità per la gestione della Startup-Sequence

Supponiamo che la :s/Startup-Sequence sia quella fornita con il sistema, e che ne esista già una alternativa,

```
:s/seqs/DoResident-Sequence
```

come avevamo detto prima. Supponiamo inoltre, che esista un file,

```
:s/seqs/current
```

il cui contenuto non sia altro che la stringa:

```
":s/seqs/Startup-Sequence.old"
```

Cioè, supponiamo di aver scritto il pathname originale associato alla startup-Sequence fornita con il sistema in un file :s/seqs/current. Si è adottata la convenzione che i file di comandi terminati con il suffisso "-Sequence" sono dei file scritti con lo scopo di implementare delle condizioni di startup.

Il modo con cui ChangeSeq lavora è descritto con il seguente pseudo-codice:

- controlla che DoResident-Sequence sia presente nella subdirectory appropriata (:s/seqs);
- recupera il pathname del file che viene attualmente usato come :s/Startup-Sequence da :s/seqs/current, e sposta il file nuovamente nella subdirectory :s/seqs (cioè, lo rinomina come :s/seqs/Startup-Sequence.old);
- rinomina DoResident-Sequence come :s/Startup-Sequence;
- scrive il pathname originale di DoResident-Sequence nel file :s/seqs/current, in modo che possa essere recuperato la prossima volta che si usa ChangeSeq.

La trasformazione di ciò nei comandi AmigaDOS è la seguente:

```
1 . ChangeSeq
2 .key FROM/a
3 .bra {
4 .ket }
5 If exists ":s/sqs/{FROM}"
6 Rename > Nil: < :s/sqs/current from
    :s/Startup-Sequence ?
7 Rename ":s/seqs/{FROM}" as
    :s/Startup-Sequence
8 Echo > :s/seqs/current "*":s/seqs/{FROM}*""
```

ChangeSeq segue strettamente il modello descritto dallo pseudo-codice. Nella linea 6, l'uso della keyword FROM è dettato solo dall'esigenza di distinguere come la linea di comando e l'input redirezionato siano associati agli argomenti di Rename; non è strettamente necessario.

La linea 8, come è stato chiarito dallo pseudo-codice, memorizza il *vero* nome della corrente startup-Sequence in :s/seqs/current. L'uso di speciali caratteri per scrivere le virgolette nello stesso file genera, nuovamente, una protezione contro l'uso dei caratteri spazio all'interno del nome del file.

ChangeSeq non è un brutto programma, ma :s/Startup-Sequence non è l'unico file che sia importante al momento del boot e che possa trarre benefici da una gestione di questo tipo. Per esempio, ci sono momenti in cui si vorrebbe che il puntatore del mouse non venisse visualizzato; si potrebbe scrivere un programma che lo trasformi in qualcosa di diverso, ma niente impedisce di scrivere un file di comandi chiamato ChangeConf che faccia pressappoco la stessa cosa.

La stessa cosa la si potrebbe dire a riguardo di qualsiasi file del tipo di :s/Shell-Startup o di file per i quali è consona una tecnica di gestione dei file di questo tipo (:devs/Mountlist, per esempio). Può comunque sembrare inutile scrivere diversi file di comandi che facciano quello che è possibile realizzare con uno solo; modifichiamo quindi ChangeSeq.

4b. ChangeBoot, un programma di utilità per la gestione dei file di boot

Possono essere fatti innumerevoli cambiamenti in ChangeBoot, ma, a scopo illustrativo, assumeremo che, per prima cosa, esistano i seguenti file:

```
:devs/confs/current
:s/seqs/current
:s/starts/current
```

Secondo, che ciascuno di questi file contenga il nome *reale* del file di boot attualmente in uso; e terzo, che le directory :devs/confs, :s/seqs e :s/starts contengano i file di boot alternativi non usati correntemente.

Ecco il listato di ChangeBoot:

```
1 . ChangeBoot
2 .key "FROM/a,
    devs/System=configuration=devs/confs/s,
    s/Startup=Sequence=s/seqs/s,
    s/Shell=startup=s/starts/s"
3 .bra {
4 .ket }
5 If exists
  :{devs/confs}{s/seqs}{s/starts}/{FROM}
6 Rename > Nil: <
    :{devs/confs}{s/seqs}{s/starts}/current from
    :{devs/System}{s/Startup}{s/Shell}-
    {configuration}{Sequence}{startup} ?
7 Rename
    :{devs/confs}{s/seqs}{s/starts}/{FROM} as
```

```
:{devs/System}{s/Startup}{s/Shell}-
  {configuration}{Sequence}{startup}
8 Echo
  >:{devs/confs}{s/seqs}{s/starts}/current "*"
  :{devs/confs}{s/seqs}{s/starts}/{FROM}*""
9 Else
10 Echo
  ":{devs/confs}{s/seqs}{s/starts}/{FROM} non
  trovato"
11 EndIf
```

ChangeBoot ha bisogno di due argomenti, in qualsiasi ordine. Il primo identifica che tipo di file di boot bisogna gestire. Il secondo è il nome del file di boot alternativo che deve essere rinominato in modo da diventare quello attuale. ChangeBoot cerca questo file in qualsiasi subdirectory implicata dal primo argomento. Se ChangeBoot non riesce a trovare il file, visualizza un messaggio. Notate che ChangeBoot compie le sue operazioni solamente sul volume corrente.

ChangeBoot sembra essere complesso perché i riferimenti ai file/directory presenti nelle sue linee cambiano in funzione di quale dei tre file di boot debba essere gestito. Dichiarando degli equivalenti di keyword (linea 2), ChangeBoot adatta le variazioni a come vengono usati realmente gli argomenti nel contesto dei singoli comandi. La restrizione sulla lunghezza delle keyword ci condiziona a rappresentare, in modo un po' deviante, dei nomi di file del tipo "devs/System-configuration" come "devs/System" - "configuration", anche se questo particolare modo di agire è utile solamente per file di boot che contengono il trattino. ChangeBoot gestisce solo un file di boot alla volta, e quindi la maggior parte dei parametri racchiusi tra parentesi vengono scaricati, lasciando solamente i parametri che hanno importanza per il file di boot in esame. Per esempio, sostituendo gli argomenti usati nella sezione 4a, le linee del programma diventerebbero:

```
5 If exists ":s/seqs/DoResident-Sequence"
6 Rename > Nil: <:s/seqs/current from
    :s/Startup-Sequence ?
7 Rename ":s/seqs/DoResident-Sequence" as
    :s/Startup-Sequence
8 Echo > :s/seqs/current
    "*":s/seqs/DoResident-Sequence*""
```

identiche alla prima versione.

La novità in ChangeBoot è che tutte le dichiarazioni di parametri sono racchiuse tra virgolette. La presenza combinata di segni di uguale ('=') e di parametri multipli richiede che l'intera specificazione venga composta in un'unica stringa. Come potremo vedere, questa stringa può racchiudere una quantità notevole di testo non legato alle keyword.

ChangeBoot ci fornisce uno strumento con il quale gestire le variazioni delle condizioni di startup; ChangeBoot, comunque, ha bisogno che il sistema sia già attivo prima che possa essere effettuata qualsiasi modifica. Questo può essere molto spesso un inconveniente. Si ha bisogno di una Startup-Sequence che permetta di scegliere tra condizioni di startup di default e condizioni specificate dalla linea di comando. Sarebbe anche di aiuto poter eseguire questa sequenza di startup in un modo di *solo default*, cioè senza dover dare l'autorizzazione all'uso di un parametro di default alla volta. Il resto di questo articolo sarà dedicato alla realizzazione di un file di comandi per la shell con il quale gestire delle condizioni durante la sequenza di startup.

5. Una sequenza di startup multiuso

La seguente sequenza di startup fà relativamente poche assunzioni sulla configurazione fisica del sistema e sul modo in cui i suoi file importanti siano distribuiti sui drive o sui volumi. Inoltre, le supposizioni che vengono fatte possono essere modificate se lo si desidera, durante lo startup, perché la sequenza viene eseguita in modo *interattivo*. Il suo progetto modulare, infine, permette di essere facilmente modificata o espansa così da avere una lista di condizioni di startup differenti come effetti della sua esecuzione nel modo di default. Questo tipo di flessibilità diventerà molto utile quando cambieranno le esigenze del proprio sistema.

Può essere interessante notare subito che la 'Startup-Sequence' che verrà proposta non è costituita da un singolo file di comandi, ma da una serie collegata di file, che iniziano con :s/Startup-Sequence sul disco di boot e finiscono (otto file più avanti) con :s/System-Sequence. Inoltre, i file iniziali e quelli finali non sono memorizzati sullo stesso volume. Come diventerà chiaro più avanti, è in virtù del suo modo di realizzazione di tipo linkato che la sequenza di startup nell'insieme è in grado di collegare dei volumi senza sapere il nome del volume stesso e di combinare modi di esecuzione di tipo interattivo e di default.

5a. Supposizioni e convenzioni

Eseguita nel modo di default, questa sequenza di startup fa uso di certe ipotesi, le quali prevedono:

1) che i file importanti per lo startup siano distribuiti su quattro dischi, i cui nomi di volume sono: Startup, Commands, Workbench e IFFfonts.

2) che il volume Startup contenga le seguenti directory: devs, l, libs ed s.

3) che il volume Commands contenga le directory c ed s.

4) che la directory :s sul volume Startup e la directory :c sul volume Commands contengano i comandi necessari per l'esecuzione dei loro rispettivi file di comandi.

5) che le directory :s sui volumi Startup e Commands contengano i file di comandi che verranno descritti più avanti.

Si ipotizza anche (sebbene i file descritti più avanti non lo richiedano) che il volume Workbench includa dei file importanti per il sistema ed eseguibili attraverso le icone come :System/Format, :Utilities/Calculator, ecc. Da notare, nuovamente, che queste ipotesi vengono fatte solamente se questi file vengono eseguiti nel loro modo di default. Il fine più importante nello sviluppo di questa sequenza è stato quello di fare in

modo che l'utente non fosse costretto a usare un particolare insieme di condizioni di startup.

Inoltre con lo scopo di dimostrare un insieme di tecniche di programmazione, la distribuzione delle directory e dei file sui dischi riflette un interesse nel come i file importanti per il sistema possano essere effettivamente compressi dentro una configurazione dell'Amiga povera di memoria. Perché compattare i file più importanti in questo modo particolare? La discussione va oltre lo scopo di questo articolo, comunque, qualsiasi lettore che si sia trovato nella condizione di controllare mezza dozzina di dischi con il semplice scopo di trovare un comune comando di AmigaDOS potrebbe avere qualche interesse alla sua pubblicazione. Quello che ci si prefigge di fare è che i lettori capiscano come questa sequenza possa essere estesa in modo da comprendere le particolari condizioni di startup legate alle loro esigenze.

5b. Principi di funzionamento

Si può pensare alla sequenza di startup come a un file di comandi che consiste in:

```
Startup:s/Startup-Sequence
Startup:s/Default-Sequence
Startup:s/Commands-Sequence
Startup:s/Shell-Sequence
Startup:s/Workbench-Sequence
Commands:s/Fonts-Sequence
Commands:s/Startup-Sequence
Commands:s/System-Sequence
```

Vedremo che la Startup:s/Shell-Sequence eseguirà la Startup:s/Default-Sequence in modo da lasciare aperto il flusso di input, cioè:

```
Execute Startup:s/Default-Sequence ?
```

Come si chiarirà più avanti, Startup:s/Default-Sequence sta aspettando uno dei due tipi di risposta:

1. il carattere 'd' oppure la stringa "default", più RETURN

2. Qualsiasi altra cosa (esclusi i caratteri speciali), più RETURN.

A seguito di una qualche risposta da parte dell'utente, il flusso di ingresso viene chiuso, e viene eseguita la parte restante di Startup:s/Default-Sequence, inclusa una chiamata a Startup:s/Workbench-Sequence. È la caratteristica principale di questa sequenza di startup nel suo insieme che il modo in cui Startup:s/Workbench-Sequence viene eseguita (con o senza l'argomento '?') dipende dalla prima risposta dell'utente al prompt della Startup:s/Default-Sequence.

Vi sembra abbastanza confuso? Riproviamo:

L'ultima istruzione nella Startup:s/Default-Sequence consiste in:

```
Execute Startup:s/Workbench-Sequence <DEFAULT$?>
```

Ora, il fatto che la Startup:s/Workbench-Sequence venga eseguita con l'argomento '?' dipende dal fatto che l'utente abbia inserito il carattere 'd' (oppure la stringa "default") come risposta al prompt della Startup:s/Default-Sequence. Se è stato usato 'd' (o "default"), la sostituzione dei parametri modifica l'ultima istruzione della Startup:s/Default-Sequence nel seguente modo:

```
Execute Startup:s/Workbench-Sequence DEFAULT
```

così la Startup:s/Workbench-Sequence viene eseguita senza l'argomento '?'; cioè il flusso di input rimane chiuso per l'utente. D'altra parte, se non fosse stata inserita sul prompt iniziale nessuna stringa idonea, l'istruzione precedente sarebbe diventata:

```
Execute Startup:s/Workbench-Sequence ?
```

così la Startup:s/Workbench-Sequence sarebbe andata alla ricerca di altro input dalla console. E' in conseguenza di questa modifica al modo di esecuzione della Startup:s/Workbench-Sequence che si ha l'opportunità, esattamente in questo punto, di effettuare una sostituzione dei parametri dalla linea di comando.

È importante notare che la Startup:s/Workbench-Sequence non fa uso della stringa "default", se non quello di passarla alla Startup:s/Commands-Sequence, la quale la passerà al file di comandi successivo e così via. Il suo unico scopo (se è stata inserita) è quello di *disattivare* l'argomento '?' e quindi di forzare l'esecuzione di default per i file di comandi successivi. Vediamo quindi come gestire gli effetti della Startup-Sequence; il principale motivo per cui l'abbiamo resa interattiva.

Oltre alla stringa di input "default" descritta prima, ogni file di comandi (successivo alla Startup:s/Default-Sequence) riceve un argomento *sostanziale*; cioè, un argomento che, se fornito, modifica l'esecuzione della startup-Sequence in modo considerevole. Come è già stato detto, questo potrebbe (ma non si limita a questo) assegnare directory logiche differenti, aggiungere quantità di buffer diverse e fare il mount di altri dispositivi. Il modo in cui queste possibilità vengano implementate sia come argomenti che come istruzioni dipende, naturalmente, da come debbano essere realizzate le condizioni di startup.

5c. Analisi dettagliata

Il primo file di comandi della sequenza di startup è, evidentemente, la Startup:s/Startup-Sequence.

```
1 ; Startup:s/Startup-Sequence
2 Run > Nil: SetAlert       ; sono delle
    tipiche istruzioni della v1.3
3 FastMemFirst              ; che possono
    avvenire subito o quando
4 BindDrivers               ; vengono estratte
    dalle directory contenute
5 AddBuffers Startup: 5     ; sul volume
```

```
    Startup. Notate che i comandi
6 FF > Nil: -0              ; inclusi su
    Startup vengono recuperati
7 SetMap usa1               ; dalla directory
    root. (non da :c)
8 Setclock > Nil: load
9 Resident CLI L:Shell-Seg system pure
10 Resident Resident pure
11 Mount NewCon:
12 NewShell NewCon:0/200/640/200/MyShell from
    Startup:s/Shell-Sequence
13 EndCLI > Nil:
```

Sulla linea 10, il comando Resident è fatto diventare residente; altri comandi lo diventeranno più avanti. Per sfruttare i comandi residenti, si deve attivare la shell avanzata (v1.3); cioè, una volta fatti eseguire comandi che sono (a) mai eseguiti più di una volta, e (b) associati con lo startup in modo invariante, si vuole uscire dal CLI ed entrare nella shell avanzata, anche se la Startup:s/Shell-Sequence non può essere eseguita (linea 12) in modo da lasciare il flusso di input aperto all'utente.

```
1 ; Startup:s/Shell-Sequence
2 Resident Startup:Execute
3 Resident Startup:CD
4 CD Ram:
5 Execute Startup:s/Default-Sequence ?
```

Startup:s/Shell-Sequence sfrutta il fatto che i comandi siano residenti per eseguire quelle poche ma basilari condizioni di startup rimaste che sono implementate prima dell'esecuzione della Startup:s/Default-Sequence; cioè, prima dell'interazione con l'utente. Una conseguenza inevitabile di ciò è la creazione, usando Execute, di file temporanei. Il motivo obbligato della rilocazione del processo in RAM: (linea 4) è quello di evitare di mettere :t nel volume Startup.

```
1 . Startup:s/Default-Sequence
2 .key "D=DEFAULT/s"
3 Execute Startup:s/Workbench-Sequence
<DEFAULT$?>
```

La ragione di esistere della Startup:s/Default-Sequence, come è già stato detto, è quella di discriminare la scelta dell'utente in modo da far eseguire (successivi) file di comandi in modo di default o in modo interattivo. Detto questo, non importa che non si abbia veramente bisogno di un file di comandi separato per dare all'utente questa possibilità; la presenza di D=DEFAULT come parametro in ciascuno dei successivi file della Startup-Sequence implica che ci sia questa possibilità a ogni prompt. D'altra parte, articolando questa possibilità di scelta in un file di comandi separato si permette la visualizzazione di un prompt descrittivo specifico del caso (vedere la sezione 5d, più avanti).

```
1 . Startup:s/Workbench-Sequence
2 .key "D=DEFAULT/s,L=LOADWB/s"
3 <LOADWB>; poichè LoadWb è sul disco di boot
    non è necessario il pathname
4 Execute Startup:s/Commands-Sequence
    <DEAFULT$?>
```

L'inserzione da parte dell'utente di "default" come argomento della linea di comando a Startup:s/Default-Sequence genera il passaggio di "default" da Startup:s/Workbench-Sequence al suo successore. La Startup:s/Workbench-Sequence, eseguita con questa modalità, toglie all'utente l'opportunità di scegliere di caricare l'ambiente del Workbench (almeno fino a che la startup-Sequence non è finita). Eseguita invece in modo interattivo, all'utente è data la possibilità di scegliere tra: 'l' o 'LoadWb' per caricare il Workbench, RETURN per passare oltre. Per molti utenti con un Amiga a corto di memoria è una domanda difficile. Essendo uno che apprezza e consiglia l'uso dell'ambiente Workbench, vorrei che non fosse così; ma se comunque la domanda deve essere posta, è meglio porla adesso, all'inizio del gioco, per risparmiare cambi di dischi inutili.

Il prossimo file di comandi è piuttosto complesso:

```
1 . Startup:s/Commands-Sequence
2 .key ",D=DEFAULTS/s"
3 "<$Commands:>c/AddBuffers" > Nil:
  "<$Commands:>" 5
4 Resident "<$Commands:>c/Assign" pure
5 Assign C:"<$Coomands:>c"
6 Resident C:Echo pure
  ;
  ; i vostri comandi preferiti
  ;
12 Resident C:MakeDir pure
13 MakeDir Ram:env
14 ; Ram:t è automaticamente creata da
  Execute Startup:s/Default-Sequence
15 Assign Env: Ram:env
16 Assign T: Ram:t
17 Resident C:Path pure
18 Path Ram: C: add
  ;
  ; altri comandi preferiti
  ;
24 Execute "<$Commands:>s/Startup-Sequence"
  <DEFAULT$?>
25 Resident Path Remove
26 Resident Assign Remove ; non necessario
  dopo che si sono realizzate le condizioni di
  startup
```

In questo file sono presenti molte cose nuove; anche se, per i nostri scopi, l'interesse principale resta l'introduzione di un parametro piuttosto inusuale: un parametro dichiarato solo in virtù della virgola con la quale esso si distingue da DEFAULT. La keyword, o luogo dove vengono mantenute tutte le modifiche sostanziali nel corrente file non è definita in funzione dell'uguaglianza di pattern; la sua unica proprietà di uguaglianza consiste nelle due stringhe che non uguaglierà: 'd' e "default". Un esame più attento del file rivela che questo parametro 'null' è presente cinque volte nel corpo del file.

La Startup:s/Commands-Sequence è memorizzata sul volume Startup. È una caratteristica del modo in cui vengono eseguiti i file di comandi il fatto che essi siano portati nello spazio di indirizzamento prima che venga effettuata la sostituzione dei parametri. Una interessante conseguenza di ciò è che, dopo aver

caricato Startup:s/Commands-Sequence, il volume Startup può essere tolto dal drive, e può essere inserito il volume Commands (o qualunque altro volume che contenga la directory alla quale c: verrà assegnata). Realizzare Startup:s/Commands-Sequence in modo che i riferimenti al volume Commands (o a un altro disco) non abbiano bisogno di essere *riempiti* prima che il volume Commands sia effettivamente richiesto implica un maggior sforzo per evitare dei ridondanti cambiamenti di disco.

Lo scopo di Startup:s/Commands è quello di rendere residenti alcuni dei comandi usati più spesso, più quelli che sono necessari per un proseguimento efficiente del resto della sequenza di startup. Non appena certi comandi vengono caricati nello spazio di indirizzamento, vengono realizzate un certo numero di condizioni di startup secondarie; per esempio, come sulla linea 18, l'effettivo percorso di ricerca per i comandi non residenti è: la corrente directory di lavoro (attualmente RAM:), poi la RAM:, poi Commands:c, infine C: (attualmente Commands:c).

La ridondanza apparente di questo path è ingannevole. Dato che Sys:System e Sys:Utilities non risiedono più sullo stesso disco, è importante che vengano aggiunti al percorso RAM: e Commands:c. Notate come, incidentalmente, il Path non possa essere fatto eseguire (Run) come un processo di background (il che, pensandoci, ha senso).

La linea 24 ipotizza che esista già una :s/Startup-Sequence al livello della root del volume al quale è assegnato c:. Questo facilita l'esecuzione di una :s/Startup-Sequence fornita con il sistema, per esempio su di un disco orientato alle applicazioni (purché questo disco possieda l'insieme di comandi della Startup:s/Commands-Sequence). Supponiamo, comunque, che c: sia assegnato a Commands:c, l'esecuzione della Commands:s/Startup-Sequence diventa:

```
1 . Commands:s/Startup-Sequence
2 .key ",D=DEFAULT/s"
3 FailAt 21
4 Assign S:"<$Commands:>s"
5 Path S: add
6 Run > Nil: Execute DoDate ; l'I/O di questo
tipo può essere concorrente
7 Execute Fonts-Sequence <DEFAULTS$?>
```

A beneficio di quegli utenti che non posseggono un clock, Commands:s/Startup-Sequence utilizza il file di comandi ausiliario DoDate, che si presume sia rintracciabile nella directory alla quale è assegnata s:; anche se non fosse così il file proseguirebbe comunque nella sua esecuzione. Dato che il volume su cui è memorizzato DoDate sarà montato durante l'esecuzione di Commands:s/Startup-Sequence, questa è una buona opportunità (meno scambi di dischi) per mandare in esecuzione un file ausiliario.

Ipotizzando, nuovamente, che l'utente abbia assegnato s: al volume Commands, ecco la Commands:s/Fonts-Sequence:

```
1 . Commands:s/Fonts-Sequence
2 .key ",D=DEFAULT/s"
3 Run > Nil: Assign Fonts:
"<$IFFfonts:>fonts"
4 Execute System-Sequence <DEFAULT$?>"
```

Fonts: è assegnato in un processo background (linea 3) in modo da non tenere ferma la Commands:s/System-Sequence.

```
1 . Commands;s/System-Sequence
2 .key ",D=DEFAULT/s"
3 FailAt 21
4 Assign Sys: "<$Workbench:>"
5 Path Sys:System Sys:Utilities Sys: add
6 ;Execute -Sequence <DEFAULT$?> ; altri file
    di comandi ?
```

In merito alla linea 3, dato che è possibile che il volume al quale l'utente assegna Sys: non contenga una o più directory fissate nella linea 5, le aggiunte fatte a Path in questa sequenza di startup sono coerenti nel contenuto, anche se non nell'ordine, con quelle della startup-Sequence *ufficiale* dell'Amiga Workbench.

5d. Come incrementare la potenza descrittiva del prompt secondario

Tutti i normali comandi dell'AmigaDOS eseguiti con l'argomento '?' inviano un template alla console per indicare all'utente che tipo di input sia richiesto. La Startup-Sequence illustrata prima non obbedisce a questa pratica in modo significativo. Supponiamo che questo sia un problema, la nostra soluzione sarà quella di usare le sequenze di controllo della console per porre e formattare del testo informativo dentro il prompt secondario e per nascondere parte della complessità del file di comandi all'utente.

Seguono ora tre esempi di startup-Sequence che utilizzano le sequenze di controllo: Per prima, Startup:s/Default-Sequence:

```
1 . Startup:s/Default-Sequence
2 .key "ESC[32m:ESC31m inserisci
  ESC[1;32mdESC[0m=ESC[1;32m DEFAULTESC[0m per
  una startup-Sequence di default | ESC[4m
  returnESC[0m per modificarlaESC,
  D=DEFAULT/s,ESC[32mESC[E "
3 Execute Startup:s/Workbench-Sequence
  <DEFAULT$?>
  ;
  ;...ecc.
  ;
```

Il prompt generato da questo file è simile a questo:

```
: inserisci d=DEFAULT per una
  startup-Sequence di default | return per
  modificarla
:
```

con i due punti, il carattere singolo 'd' e la stringa "DEFAULT" stampati con il colore 2. La parte funzionale della dichiarazione, essendo stampata con il colore 0, non viene visualizzata.

Come seconda, Startup:s/Workbench-Sequence:

```
1 . Startup:s/Workbench-Sequence
2 .key "ESC[32m:ESC[31m inserisci
    ESC[1;32mlESC[0m=ESC[1;32mLoadWbESC[0m per
    caricare l'ambiente Workbench |
    ESC[4mreturnESC[0m per il solo CLI
    ESC,D=DEFAULT/s,L=LOADWB/s,ESC[32ESC[E "
3 <LOADWB>
4 Execute Startup:s/Commands-Sequence
    <DEFAULT$?>
  ;
  ;...ecc.
```

dove il prompt somiglia a:

```
: inserisci l=LoadWb per caricare l'ambiente
   Worbench | return per il solo CLI
:
```

L'ultimo esempio di una integrazione di una sequenza di controllo è Startup:s/Commands-Sequence

```
1 . Startup:s/Commands-Sequence
2 .key "ESC,ESC[32m;ESC[31m inserisci
    ESC[3;32mroot:ESC[0m per assegnare C: al
    volume ESC[3mroot:ESC[0mc | ESC[4mreturn
    ESC[0m per
    ESC[1;3mCommands:ESC[0mcESC,D=DEFAULT/s,
    ESC[32mESC[E "
3 "<ESC$Commands:>c/AddBuffers" > Nil:
"<ESC$Commands:>c" 5
4 Resident "<ESC$Commands:>c/SAssign" pure"
  ;
  ; ...altri comandi
  ;
24 Execute "ESC$Commands:>s/Startup-Sequence"
    <DEFAULT$?>
  ;
  ;...ecc.
  ;
```

per la quale il prompt consiste in:

```
: inserisci root: per assegnare C: al volume
root:c | return per Commands:c

:
```

nel quale i due punti e la prima delle due stringhe "root" sono stampati con il colore 2.

Nella sua versione iniziale (cioè senza sequenze di controllo), questo file di comandi (e quelli precedenti) associavano gli argomenti della linea di comando diversi da 'd' e da "default" con un parametro null. Sopra, questo parametro null è stato trasformato in una serie di caratteri non stampabili che comprendono la stessa sequenza di controllo. Cioè, il posto dove vengono mantenuti gli argomenti essenziali della linea di comando è la stringa:

```
ESC
```

Dopo tutto, chi ha detto che le keyword dei file di comandi debbano essere dei caratteri stampabili?

5e. DoDate - un programma di utilità per gestire la data

DoDate compare nella precedente Commands:s/Startup-Sequence.

```
1 . DoDate
2 .bra {
3 .ket }
4 FailAt 21
5 Date > Nil: < "NewCon:5/90/629/25/DoDate:
    inserisci la data (GG-MM-AA) e/o l'ora
    (OO:MM); [return] per finire." ?
6 If fail
7   Execute DoDate
8 EndIf
```

DoDate usa la ricorsione fino a quando l'utente non fornisce un argomento corretto al comando Date.

5f. Sommario

Se questa startup-Sequence non fosse interattiva, sarebbe comunque composta da molte istruzioni. Modificare l'ordine delle istruzioni nei file per sfruttare la residenza dei comandi riduce, ma non elimina, l'investimento in tempo/energia da parte dell'utente. Gli utenti che passano la maggior parte del loro tempo con una singola applicazione certamente penseranno che un sforzo di questa portata non abbia alcun senso. Lo stesso lo si potrebbe dire di utenti che, avendo la necessità di memoria contigua, o per semplice abitudine, rieseguono il boot dopo essere usciti da qualsiasi applicativo.

D'altra parte, gli utenti che devono gestire delle condizioni di sistema prima di far partire le proprie applicazioni hanno tipi di utilizzo diversi e possono trovare interessanti alcuni aspetti di questo modello. L'estensione delle tecniche qui illustrate per creare un albero decisionale delle condizioni di startup non è dimostrata ma è abbastanza intuibile.

I programmi della shell hanno un alto overhead e un campo di azione limitato. Se un programma di shell abbia senso come approccio ad un particolare problema di programmazione dipende dal tipo di uso che uno fa del computer e dal livello di astrazione del problema. Dove queste considerazioni siano rivolte alla vitalità della programmazione di shell, i programmi di shell sono di gran lunga più facili da realizzare che i loro equivalenti scritti in C o in BASIC. Il loro uso richiede una certa comprensione dei concetti legati all'AmigaDOS e alla disponibilità di esempi. Questo articolo è mirato alla divulgazione dell'ultimo di questi prerequisiti.

"LUCAS" per Amiga 1000

Una scheda acceleratore con 68020/68881

di Brad Fowles

Brad Fowles è un progettista della Anakin Research ed è il responsabile della parte hardware della tavoletta grafica Easyl prodotta dalla Anakin. Brad si è attivamente interessato ad Amiga sin dal lancio della macchina nel 1985.

Sapete senz'altro tutti quanto software di alta qualità sia disponibile nel pubblico dominio. Come appassionato di hardware, non ho potuto fare a meno di invidiare la facilità con la quale il software può essere distribuito tra sviluppatori e utenti. La circolazione delle idee attraverso le BBS costituisce senz'altro un vantaggio per tutti. Gli sviluppatori di hardware, invece, sono costretti a condurre un'esistenza che al confronto appare solitaria; lo scambio di idee e di tecniche è nel loro caso resa difficoltosa da problemi economici e logistici.

Ha senso allora parlare di hardware di pubblico dominio? Certamente nessuno può mettersi a distribuire circuiti stampati gratuitamente, ma si può invece fare una cosa molto simile: rendere disponibile quanta più informazione è possibile e produrre e distribuire circuiti stampati al puro costo di produzione.

Il progetto illustrato in questo articolo è una scheda detta LUCAS (da Little Ugly Cheap Accelerator System, cioè orribile piccola scheda acceleratrice economica...) che va a sostituire il 68000 in un Amiga 1000. La scheda, che permette un aumento considerevole delle prestazioni, consente l'impiego del coprocessore matematico 68881 ed è il primo passo per arrivare all'uso di memoria a parole di 32 bit.

Sulla scheda sono presenti un 68020 e un 68881 che funzionano a 16 MHz e tutta la circuiteria di interfaccia (che consiste di quattro PAL, quattro componenti discreti, un quarzo da 16 MHz, due reti resistive SIP e alcuni condensatori) necessaria per adattare i cicli del 68020 a quelli del 68000. LUCAS presenta anche un connettore che permette, in un secondo tempo, di aggiungere memoria a 32 bit. Detto tra noi cercherò di convincere gli editori di Transactor a pubblicare il progetto di un'espansione di memoria per questo sistema tra qualche mese.

Il solo circuito stampato è disponibile presso il sottoscritto al costo di $40; il set di quattro circuiti PAL invece costa $25 (le modalità di ordine sono chiarite al termine di questo articolo). Il resto dei componenti dovrebbe essere facilmente reperibile presso i vostri rivenditori locali. Nell'articolo sono mostrati lo schema e le equazioni dei PAL. Chiunque voglia adattare la scheda per l'uso con un Amiga 500 non deve far altro che rivolgersi a me e sarò felice di fornirgli il materiale necessario. Il progetto del circuito stampato è stato fatto con il programma P-Cad su un... ehm! ...un AT.

Se possedete un Amiga 1000 e volete sperimentare l'aumento di prestazioni dato dall'accoppiata 68020/68881, questa è forse la maniera più a buon mercato. Tenete presente, però, che processore e coprocessore, purtroppo, costano in tutto circa $370 (canadesi). Il mio obbiettivo è di rendere il resto il più economico possibile. La scheda nella sua totalità dovrebbe venire a costarvi non più di $475 e senz'altro qualcosa in meno in zone ben fornite.

Ho intrapreso questo progetto per tre ragioni. La prima è che volevo una scheda simile e non potevo permettermi una versione commerciale. La seconda è che ho diversi amici che usano Sculpt 3D e Animate della Byte by Byte (disponibili entrambi nella versione per 68020/68881) che sentivano la necessità di maggiore potenza di calcolo per produrre immagini a velocità sufficiente da poterci guadagnare soldi. In terzo luogo, ho pensato che tutti i possessori di Amiga 1000 degni del nome di hacker dovessero aver bisogno di un modo per iniziare ad esplorare il futuro di Amiga.

Quando ho iniziato il progetto, ho inizialmente fatto riferimento a un articolo tratto da EDN, 9 gennaio 1986, pagg. 216-219. In seguito, sono venuto in possesso della "application note" della Motorola AN944/D, MC68020 and MC68881 Platform Board for Evaluation in 16-bit System. Se volete comprendere meglio il funzionamento di questa scheda, vi raccomando caldamente questi due riferimenti, in particolar modo l'ultimo. Sfortunamente è impossibile, in un articolo come questo, dare qualcosa di più di una sommaria descrizione del funzionamento della scheda. Nella parte tecnica di questo articolo, cercherò di mettere in evidenza gli aspetti del progetto che sono specifici di Amiga, ma a ogni modo un'approfondita comprensione vi richiederà ulteriori investigazioni e ricerche. A questo proposito, vi raccomando anche i manuali utente per il 68020 e il 68881 editi dalla Motorola.

Okay, siamo arrivati alle "condizioni di garanzia": se vi procurate una basetta, assemblate tutto correttamente e installate la

scheda nel vostro Amiga, non dovreste avere alcun problema e nel giro di una o due sere dovreste avercela fatta. Se, però, incontrate dei problemi, sta alla vostra ingegnosità e alle vostre capacità tecniche risolverli. Se non avete esperienza nell'uso del saldatore, per favore, non affrontate questo progetto. Da parte mia, tenterò di dare una mano a chiunque trovi delle difficoltà. Ci sono tre modi per mettersi in contatto con me: con USENET ad anakin@gpu.utcs,toronto.edu (Brad Fowles), con BIX nell'area ANAKIN o AMIGA oppure con la posta tramite Transactor. Spero, comunque, che ci sia un interesse sufficiente a far sì che altri si aggiungano a me in questa opera di assistenza tecnica. Non c'è da parte mia alcuna obiezione se qualcuno vuole produrre i circuiti stampati, montare i componenti e installare le schede ad un prezzo modesto, ma vi prego di ricordare che lo scopo di tutto questo deve rimanere quello di rendere disponibili le schede all'utente finale al minor prezzo possibile. Se non vi ho convinto a desistere dall'intraprendere questo progetto, potete continuare la lettura di questo articolo; in caso contrario, arrivederci e grazie dell'attenzione.

Una volta che avete ottenuto la scheda vuota e vi siete procurati tutti i componenti, seguite le istruzioni che accompagnano le basette e con attenzione saldate tutti gli zoccoli per i circuiti integrati e per il quarzo; saldate, quindi, le reti resistive e i condensatori. Inserite la speciale prolunga nello zoccolo per il 68000 e saldatevela dentro.

L'installazione è molto semplice, ma deve essere fatta con cura. Per prima cosa rimuovete la copertura di plastica e lo schermo EMI dall'unità base di Amiga. Sulla parte sinistra del circuito stampato, proprio accanto al connettore per le espansioni, troverete il 68000. Estraetelo delicatamente dallo zoccolo e mettetelo in un posto sicuro. Inserite ora la scheda LUCAS nello zoccolo del 68000, facendo attenzione a verificare che tutti e 64 i pin si inseriscano correttamente nello zoccolo. Se volete fare il lavoro in maniera più comoda e sicura, rimuovete il drive interno in modo da vedere meglio nella zona delle operazioni. State però attenti alla piattina del drive, perché pieghe strane possono causare noiosi problemi. Comunque, se lavorate con cura e non forzate nulla, non dovreste avere grossi problemi. Potete fare delle prove preliminari "a cuore aperto" e una volta soddisfatti, montare nuovamente lo schermo EMI e il coperchio di plastica. Questo è tutto. Il vostro cuore può ora tornare a battere con il ritmo consueto.

Non c'è bisogno che sappiate molto del funzionamento interno della scheda LUCAS per divertirvi a usarla; comunque, per tutti coloro i quali vorrebbero sapere qualcosa in più del funzionamento di un 68020, che lavora a 32 bit, in una macchina progettata per un 68000, che invece lavora a 16 bit, alla fine di questo articolo c'è una spiegazione piuttosto dettagliata degli aspetti tecnici fondamentali della scheda LUCAS.

I benchmark

Per darvi un'idea del miglioramento delle prestazioni che si ottiene con la coppia 68020/68881, ho condotto delle prove con quattro programmi che mi sono stati dati alla Conferenza degli Sviluppatori tenutasi a Washington. I "benchmark" che mi accingo a presentare sono stati ottenuti su un Amiga 1000 con un'espansione di memoria da 2 Mega della Microbotics e un hard disk da 20 Mega della Comspec. Il sistema operativo usato è il KickStart 1.21 con il Workbench 1.3 Gamma 7. Va osservato che quando si installano il 68020 e il 68881, le nuove library matematiche IEEE che usano il coprocessore vengono impiegate in maniera del tutto trasparente. Le prove si riferiscono a un sistema con il 68000 e a un sistema con la LUCAS.

```
Savage:
     68000 470.0 sec. Errore: -6.9e-7
     LUCAS 14.5 sec . Errore: -5.7e-4

Whetstone:
     68000 24 Kwhets/sec
     UCAS 126 Kwhets/sec

Calcpi:
     68000 4.85 Kflops/sec Errore: -1.39e-11
     LUCAS 11.90 Kflops/sec Errore: -2.78e-11

Float:
     68000 10000 iterazioni 45.74 sec
           256000 iterazioni 286.96 sec
     LUCAS 10000 iterazioni 12.80 sec
           256000 iterazioni 118.56 sec
```

La velocità, ovviamente, potrebbe essere ulteriormente aumentata impiegando le istruzioni "F" per le operazioni in virgola mobile e usando RAM statica a 32 bit e no-wait-state. A conclusione di questo paragrafo sui "benchmark" ricordate che i "benchmark" sono come i discorsi dei politici: hanno un senso solo in apparenza.

Alcune considerazioni sul software

La maggior parte del software gira senza problemi col 68020, però ce ne sono alcuni che vanno inevitabilmente in crash. Una delle principali cause è data dal fatto che l'istruzione MOVE SR,<ea> è implementata in maniera differente nei due microprocessori. Nel 68000 questa è un'istruzione che può essere eseguita in "user mode"; nel 68020, come nel 68010, questa stessa istruzione può essere eseguita solamente nel "supervisor mode". Il risultato di tutto questo è che la sua esecuzione in un Amiga con 68020 conduce direttamente a una violazione di privilegio, che viene "punita" con un "guru meditation".

Se scrivete software, evitate dunque questa istruzione. Al suo posto usate la funzione di library GetCC(), che sul 68020 si traduce nell'istruzione MOVE CC,<ea> che in user mode è perfettamente lecita. Per un 68000, invece, la funzione si traduce nell'istruzione MOVE SR,a; in questa maniera siete al sicuro in entrambi i casi. Per far girare sul 68020 quei programmi che si rifiutano di funzionare per la ragione che ho appena spiegato, si può lanciare prima un programma chiamato DeciGel che intercetta l'istruzione incriminata e opera in maniera corretta. DeciGel è reperibile sia nel disco che accompagna questo numero di Transactor sia nel Fish #18.

Se fate parte di quelle persone che ritenevano un'idea astuta mettere dell'informazione negli otto bit superiori di un indirizzo, sappiate che è arrivato il momento di pagare il fio dei vostri misfatti.

Sappiate anche che se nel programma che scrivete usate istruzioni specifiche del 68020, lo stesso programma non funzionerà mai su un Amiga standard. Per ulteriori chiarimenti, potete consultare il capitolo 21 delle Note della Washington Amiga Developer Conference, che ha per titolo "Software Issues in 32-bit Amiga Systems".

La versione 1.3 del software di sistema comprende nuove librerie matematiche in doppia precisione IEEE, che sono in grado di sfruttare la presenza del 68020 e del 68881 e che quindi rendono notevolmente più veloce qualsiasi programma che usi le librerie matematiche.

Se volete nei vostri programmi operazioni in virgola mobile a una velocità strabiliante, la cosa migliore da fare è di ricompilare il sorgente in modo che vengano usate le istruzioni "F". Nel disco che vi sarà recapitato insieme con la basetta, troverete i due programmi Mandslow e Mandfast. Questi programmi derivano dal famoso programma di immagini di Mandelbrot scritto da R.J. Mical e sono del tutto uguali, eccezion fatta per un particolare: Mandslow è stato compilato per funzionare su un Amiga standard, mentre Mandfast fa largo uso delle istruzioni "F" e quindi del coprocessore. Questa sola differenza fa sì che un'immagine di Mandelbrot che su un Amiga standard viene generata in un'ora e 20 minuti richiede solo 4 minuti e 20 secondi con la scheda LUCAS installata.

Compatibilità

La scheda LUCAS funziona perfettamente con tutte le schede di espansione che ho, ma sicuramente ce ne sarà in giro qualcuna con cui si verificano dei problemi. A tal proposito, ho intenzione di mantenere una lista delle schede che risultano compatibili e di quelle che al contrario si rifiutano di funzionare con la LUCAS e di aggiornarla periodicamente su Usenet e su BIX. Quelle con cui è stata provata con esito favorevole sono il controller hard disk della Comspec, l'espansione di memoria da 2 Mega sempre della Comspec, la tavoletta grafica Easyl e l'espansione di memoria StarBoard da 2 Mega della Microbotics.

La scheda, poi, funziona piuttosto bene anche a 20 MHz, ma di tanto in tanto si inchioda. Del resto, io uso componenti da 16 MHz; quando tento di capire dove si è originato il guasto, trovo che il problema nasce probabilmente nella memoria cache per le istruzioni presente nel chip stesso del 68020. Se disponete di componenti da 20 MHz, provateci e poi fatemi sapere; anche se avete componenti da 16 MHz, il piacere di provare ripaga ampiamente il costo di un quarzo da 20 MHz. E poi, chi lo sa? Potreste anche essere fortunati...

Conclusioni

Le prestazioni di un Amiga 1000 con la scheda LUCAS migliorano notevolmente, ma è comunque inutile aspettarsi miracoli. Nel caso di elaborazioni generiche come le compilazioni, l'aumento di velocità si aggira su una volta e mezzo, il che forse non giustifica neanche l'affanno. I vantaggi si sentono molto in applicazioni che fanno un uso intensivo delle operazioni in virgola mobile e soprattutto quando i programmi contengono istruzioni "F" che usano direttamente il coprocessore.

D'altro canto, la scheda consente l'uso di memoria a 32 bit e questa può consentire considerevoli aumenti di prestazioni anche in casi generali. Prevedo di progettare due espansioni: una che impieghi RAM dinamiche da 100 o 120 ns e un'altra con RAM statica ad alta velocità per operare a 16 MHz in no-wait-state. Benchè sappia perfettamente che la maggior parte dell'aumento di velocità deriva, in effetti, dall'uso di memoria a 32 bit, vi confesso che non resisto alla curiosità di vedere che cosa succede a 16 MHz in no-wait-state.

Brad fornisce il circuito stampato e i chip PAL per questo progetto a un prezzo che permette appena di coprire i costi: $40 per la scheda (e un disco) e $25 per i quattro PAL. Gli ordini vanno inviati al seguente indirizzo:

RR #5 Caledon East
Ontario, Canada
L0N 1E0

Descrizione tecnica

Con la scheda LUCAS installata, il sistema può essere visto come diviso in due blocchi funzionali; di questi, uno lavora apparentemente alla frequenza di 7.16 MHz e in maniera sincrona con i chip dedicati al video o ad altre funzioni speciali mentre l'altro risulta un sistema asincrono da 16 MHz costituito dal 68020, dal coprocessore 68881 e dalla memoria a 32 bit eventualmente presente sul bus della LUCAS.

Gli obbiettivi fondamentali di questo progetto sono di rendere possibile il funzionamento in maniera asincrona rispetto al clock di Amiga tutte le volte che ciò risulta possibile (in modo tale da ottenere frequenze di clock di 16 MHz o addirittura più elevate) e di servirsi unicamente dei segnali presenti sullo zoccolo del 68000 per ottenere tutti quelli necessari al funzionamento della LUCAS (così che l'installazione risulti estremamente facile).

Per soddisfare questi requisiti, la scheda deve comportarsi come un 68000 con clock a 7.16 MHz quando hanno luogo cicli macchina che coinvolgono l'uso del bus di Amiga; quando, però, la scheda compie elaborazioni interne o dialoga con il 68881 o con la memoria a 32 bit, il clock deve portarsi a 16 MHz e il ciclo macchina di accesso al bus deve avvenire in tre impulsi di clock, al posto dei 4 che il 68000, invece, impiega.

Il 90% dei problemi incontrati nello sviluppo di questa scheda sono stati relativi proprio a far sì che il 68020 si comporti come il 68000 che va a sostituire quando deve interagire con l'architettura di Amiga e che poi parta come una scheggia non appena gli è possibile.

Le linee dei dati e degli indirizzi non hanno presentato alcun problema, essendo state riportate pari pari dal 68020 allo zoccolo del 68000. Notate che le 16 linee di dati sono state collegate alle linee dati D16-D31 del 68020 e che gli otto bit superiori del bus degli indirizzi sono stati lasciati scollegati.

Nel prosieguo dell'articolo ho usato il simbolo "*" per indicare

segnali attivi bassi e quindi *AS significa che AS è vero quando il suo livello logico è basso. Le equazioni dei PAL sono scritte in formato CUPL e di questo chiedo venia a tutti gli utenti di PALASM.

L'interfaccia 68020-68000

Il 68000 ha un bus con struttura asincrona. Il processore manda basso *AS, cioè rende attivo Address Strobe, per iniziare un ciclo di bus e attende l'attivazione di *DTACK per terminarlo. Tutto ciò ha luogo generalmente in 4 o 6 cicli, ma può essere prolungato da una periferica che lo ritenga necessario. Il 68020 funziona in maniera molto simile; l'unica differenza è che ci sono due segnali distinti al posto di *DTACK, chiamati *DSACK0 e *DSACK1. Poiché il 68020 può indirizzare byte (8 bit), word (16 bit) e longword (32 bit), deve disporre di qualche modo per distinguere i vari casi. Ciò è ottenuto tramite il dimensionamento dinamico del bus. Una periferica risponde a un ciclo di bus, attivando i segnali *DSACKx in maniera da specificare al 68020 il tipo di trasferimento.

DSACK0	DSACK1	TRANSFER SIZE
0	0	32 bit
1	0	16 bit
0	1	8 bit
1	1	stato di attesa

I cicli di bus su Amiga coinvolgono sempre 16 bit, per cui come risposta nel caso di accesso a risorse di Amiga si attiva sempre solo *DSACK1. Se invece il ciclo di bus intende accedere al 68881 o alla memoria sul bus della LUCAS, va attivata un'opportuna combinazione dei segnali *DSACKx.

In generale, quando non vengono inseriti stati di attesa, cioè operando in no-wait-state, il 68000 effettua un ciclo sul bus in quattro impulsi di clock; il 68020, invece, ne impiega solo tre. Per correggere questa disparità, è necessario ritardare l'attivazione di *AS e di *DS (Data Strobe, linea di abilitazione dei dati) fino al fronte di salita della fase S2 del clock a 7.16 MHz della CPU. A tal scopo sono impiegati i flip-flop U8a e U8b: infatti, applicando il segnale *AS proveniente dal 68020, dopo averlo invertito, all'ingresso di reset del del flip-flop e prelevando il segnale *AS da mandare ad Amiga dall'uscita invertita del flip-flop si ottiene proprio il ritardo desiderato e si attiva il segnale interno di temporizzazione *AS20DLY in maniera corretta. La stessa tecnica viene impiegata nel caso di *DS.

L'indirizzamento del singolo byte viene ottenuto nel 68000 tramite i segnali Upper Data Strobe (*UDS) e Lower Data Strobe (*LDS). Nel 68020, al contrario, c'è un unico segnale (*DS) e per individuare il tipo di trasferimento vengono presi in considerazione i segnali *DSACK0 e *DSACK1 e le prime due linee del bus degli indirizzi A0 e A1. Notate che i byte appaiono sulle linee dalla 24 alla 31 del bus dei dati, mentre le word sulle linee dalla 16 alla 31. E' pertanto necessario creare *UDS e *LDS; per far questo è sufficiente scrivere le seguenti equazioni per i PAL:

```
!UDS=(!DS20DLY)&(!A0)&(CPCS)
!LDS=(!DS20DLY)&(SIZ1)&(CPCS)
(!DS20DLY)&(!SIZ0)&(CPCS)
(!DS20DLY)&(A0)&(CPCS)
```

Nota: le linee di controllo dei dati *UDS e *LDS non sono attive durante i cicli dedicati al coprocessore (CPCS)

Il 68000 contiene al suo interno la logica necessaria per essere compatibile con i prodotti della serie 6800. Poichè la presenza di questa circuiteria viene sfruttata in Amiga per lavorare con gli 8520, dobbiamo riprodurre sulla scheda questa caratteristica in qualche modo, dal momento che il 68020 ne è sprovvisto. In aggiunta, dobbiamo generare un segnale secondario di clock detto E e che ha frequenza pari a 1/10 di quella del clock della CPU e un duty cycle di 60% basso e 40% alto. Tale segnale viene ottenuto per mezzo di un contatore modulo 10 nel PAL U4. Quando viene eseguito un ciclo compatibile con la famiglia 6800, Amiga o una periferica genera il segnale *VPA (Valid Peripheral Address, cioè indirizzo della periferica valido). Quando ciò accade il 68000 si sincronizza con il clock E, attiva il segnale *VMA (Valid Memory Address, cioè indirizzo della memoria valido) e termina il ciclo sul fronte di discesa del clock E. L'equazione

```
!Z3=!QD#
QC#
QB#
QA;
```

sul PAL U4 in combinazione con l'equazione

```
!Z1.D=(DS20DLY)&(!Z1)#
(DS20DLY)&(Z3)&(!VMA);
```

attiva *DSACK1 al nono stato del clock E tramite la generazione del segnale Z1 in modo tale che il ciclo "lungo" possa avere regolarmente termine.

L'interfaccia 68020-68881

Il segnale di selezione dipositivo (*CS, da Chip Select) del MC68881 deve essere ottenuto dal 68020. Il 68020, quando vuole chiamare in causa il coprocessore, genera un 111 sulle linee Function Code e un 0010 sulle linee del bus degli indirizzi A13-A15. Essendoci un solo coprocessore in questo progetto, A13-A15 non vengono prese in considerazione. Il resto va a generare CPCS, usato come *CS del 68881 nella seguente maniera:

```
CPCS=(FC2)&(FC1)&(FC0)&(!A19)&(!A18)&(!A17)&
(!A16)
```

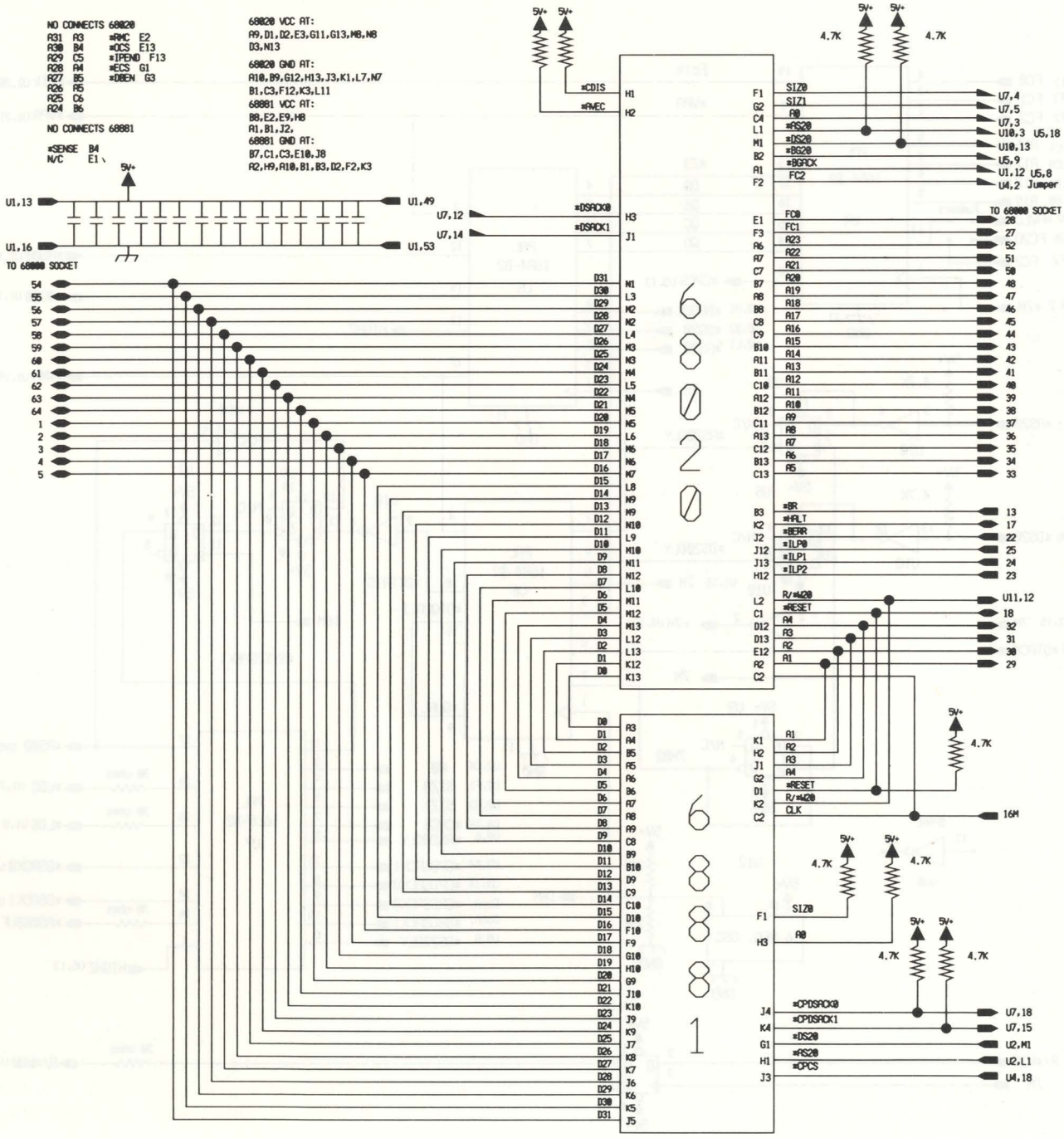
Lo Zen e l'arte della terminazione del ciclo

La generazione del segnale *DSACK1 dal *DTACK di Amiga mi ha portato più volte a dubitare non solo della mia salute mentale, ma anche della validità dei principi di logica ai quali l'universo ubbidisce. Il segnale *DTACK da Amiga dovrebbe apparire ed essere letto durante la fase S4 del ciclo di clock. Invece non è così; quando si accede alla memoria interna le cose vanno più o meno come previsto, ma quando si accede alla memoria esterna (fast memory) *DTACK torna indietro quasi immediatamente. Il fatto è particolarmente importante dal

momento che *DTACK è il solo modo che abbiamo per determinare la lunghezza di un ciclo. Tra un momento vedremo come trattare opportunamente questa anomalia.

Il 68020 opera a 16 MHz, quindi in maniera del tutto asincrona rispetto al clock di Amiga; in qualche istante, però, dovremo ad ogni modo sincronizzarci con il clock a 7.16 MHz di Amiga. Il momento ideale per farlo è quando le due linee di clock C1 e C3 sono nello stato di C1 alto e C3 basso. Purtroppo questi due segnali non arrivano alla CPU e quindi non possono essere presi direttamente dallo zoccolo, come d'altro canto intendiamo fare. Possiamo, però, accontentarci del segnale di clock a 7.16 MHz, che ovviamente arriva al 68000, e ottenere da questo un segnale in qualche maniera utile. Ciò che infatti viene fatto sulla LUCAS è di dividere per due il clock a 7.16 MHz e di farne poi un OR logico con il segnale di clock originale; quanto ottenuto si comporta nella stessa maniera di C1 alto e C3 basso (solo a questo punto tornai ad avere fede nei principi di logica dell'universo). Nelle equazioni PAL questo segnale compare con il nome di DTPRELIM (DTack PRELIMinary). Il problema del reperimento del momento opportuno per risincronizzarsi si può ora dire risolto.

In un'architettura meno bizzarra, la sola combinazione dei segnali *DTACK e Z1 (quello impiegato per la terminazione dei cicli VMA, VPA) sarebbe stata sufficiente per ottenere il termine SYSDSPRE1 (SYStem DSack1 PREliminary 1), ma nel nostro caso dobbiamo attendere fino a che *DTPRELIM diventi vero per sincronizzarci correttamente con Amiga e in più dobbiamo fare attenzione all'anomala risposta rapida di *DTACK qualora si acceda alla FAST ram.



Confusi? Non abbiate paura! Le cose migliorano.

La maggior parte delle espansioni di memoria dinamiche, quando sono connesse al bus di Amiga, mandano alto XRDY per evitare l'attivazione di *DTACK mentre sono impegnate in un ciclo di refresh, inserendo in questo modo un numero sufficiente di stati di attesa. Il problema a questo punto è che, non appena XRDY viene attivato, si verifica un impulso spurio di 20-30 ns su *DTACK che porta il 68020 a terminare il ciclo prima che i dati abbiano perfino l'intenzione di arrivare sul bus. Fortunatamente una soluzione c'è e consiste nell'evitare di lasciar passare l'impulso su *DTACK fino alla fase S4 del clock di Amiga. A tal fine ritardo *AS20DLY una volta per *DTQUAL e un'altra per *DTQUAL1. DTQUAL va quindi a far parte del termine in cui compare *DTACK, mentre *DTQUAL1 è collegato a QUAL, che a sua volta è aggiunto al termine di *DTACK; in termini di equazioni PAL, questo corrisponde a:

```
!SYSDSPRE1=!Z1#
(!DTACK)&(!AS20DLY)&(!DTQUAL)&(!QUAL)
```

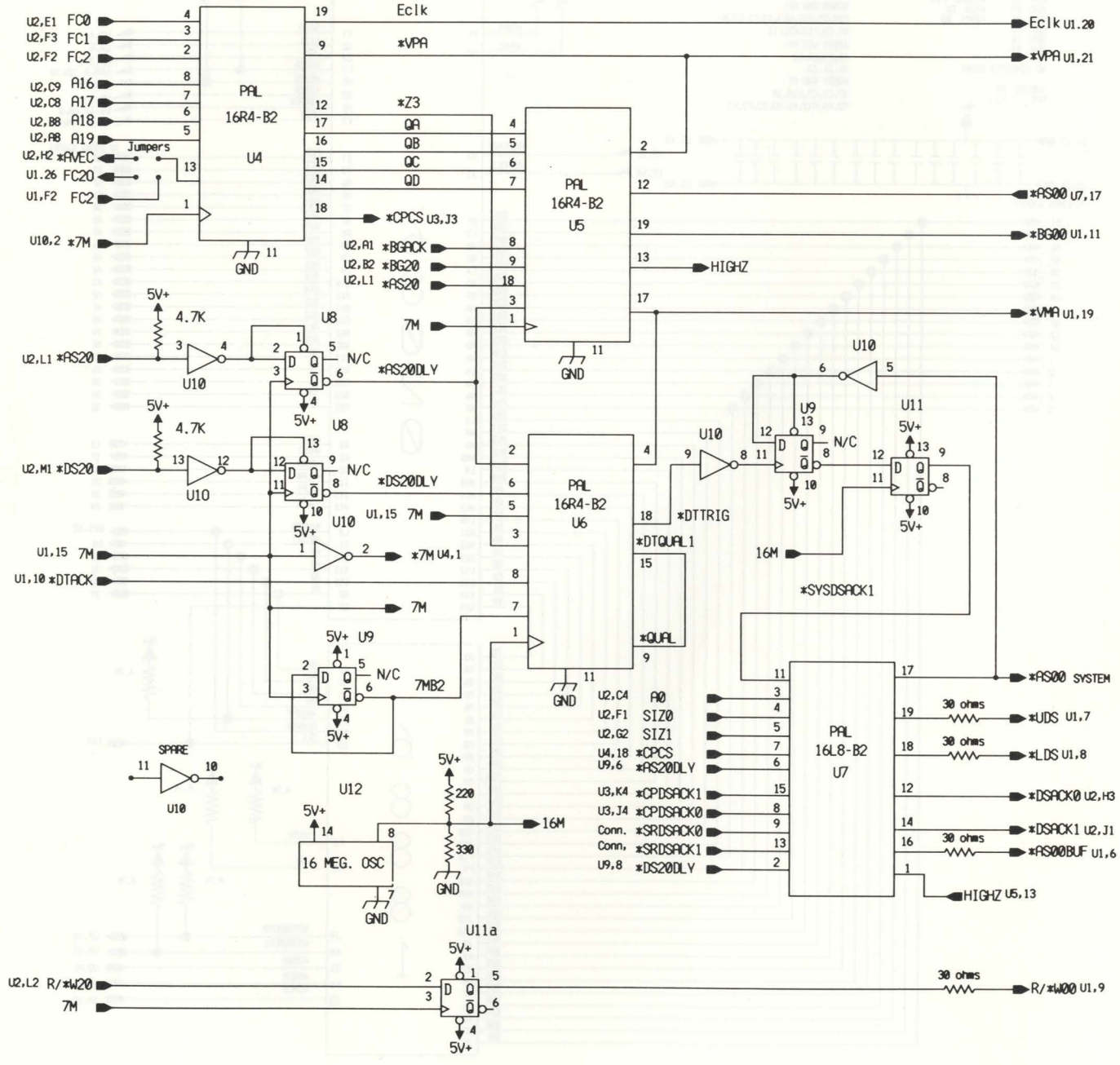
Questo basta per risolvere il problema di *DTACK. Il segnale ottenuto viene poi ritardato di altri 7 ns con:

```
!BUFOUT=!SYSDSPRE1
```

e sincronizzato con Amiga con:

```
!DTTRIG=(!BUFOUT)&(!DTPRELIM)
```

Abbiamo così un fronte sincrono con i 7.16 MHz di Amiga;



con questo fronte pilotiamo un flip-flop che ha pazientemente atteso che tutto questo pandemonio terminasse e che, a sua volta, invia il segnale ad un ulteriore flip-flop per renderlo sincrono con il clock a 16 MHz. Il segnale *SYSDSACK1 ora ottenuto viene in seguito immesso nel PAL U7 per le altre necessarie elaborazioni.

Abbiamo quasi finito. Rimane solo da svelare la funzione del PAL U7. Questo integrato tra le altre cose combina *SYSDSACK1 con il *DSACK1 del 68881, con *CPDSACK1 e con *SRDSACK1 per ottenere trionfalmente *DSACK1.

Che cosa c'è di più semplice? *DSACK è generato solamente dal 68881 e dalla futura RAM statica.

L'Assegnamento del Bus

La tecnica di assegnamento del bus ricalca quella del 68000 con un'eccezione. Durante i cicli impiegati dal coprocessore, il segnale *AS proveniente dal bus del 68000 viene bloccato; questa circostanza, purtroppo, può dar luogo a problemi. Se il 68020 comincia un ciclo di coprocessore con *AS bloccato e risponde a una richiesta del bus, segnalata da *BR (Bus Request), con la cessione del bus, attivando cioè *BG (Bus Grant), lo stesso 68020 poi si aspetterà che il richiedente del bus attenda la disattivazione di *AS. *AS, però, è bloccato e pertanto già disattivato: il risultato è un uso improprio del bus. Per evitarlo, bisogna impedire l'attivazione di *BG fino a quando l'interfaccia non disattiva *AS. La seguente equazione PAL si occupa proprio di questo:

```
!BG00=(!BG20)&(!Z2)&(AS20)
```

E con questo abbiamo veramente finito.

Equazioni dei quattro chip PAL in formato CUPL

```
PARTNOU4 ;
NAMEFinalu4 ;
REV03 ;
DATE21 Gennaio 1989 ;
DESIGNERBrad Fowles ;
COMPANYAnakin ;
ASSEMBLYLucas ;
LOCATION U4 ;
/* PAL16R4B2 */
/* SPECIFICHE DEL PAL */
/* INTERFACCIA PER AMIGA 68020-68881/68000 */
PIN 1 = 7MN ;
PIN 2 = FC2I ;
PIN 3 = FC1 ;
PIN 4 = FC0 ;
PIN 5 = A19 ;
PIN 6 = A18 ;
PIN 7 = A17 ;
PIN 8 = A16 ;
PIN 9 = VPA ;
PIN 12 = Z3 ;
PIN 13 = FC20 ;
PIN 14 = QD ;
PIN 15 = QC ;
PIN 16 = QB ;
PIN 17 = QA ;
PIN 18 = CPCS ;
PIN 19 = E ;

!E = (!QD )& (!QC )#
     (!QD )& (!QB );

!CPCS = (FC2I )& (FC1 )& (FC0 )& (!A19 )& (!A18 )&
        (!A17 )& (!A16 );

!FC20 = !FC21 ;

!Z3 = !QD #
       QC #
       QB #
       QA ;

!QA.D = QA ;

!QB.D = (!QB ) & (!QA )#
        ( QB ) & ( QA )#
        ( QD );

!QC.D = (!QC ) & (!QA )#
        (!QC ) & (!QB )#
        ( QC ) & ( QB ) & ( QA );

!QD.D = (!QC ) & (!QD )#
        (!QB ) & ( QA )#
        (!QD ) & (!QA );
/*
DESCRIZIONE: CONTATORE PER DIECI, CLOCK 6800, LOGICA
             PER LA SELEZIONE DEL COPROCESSORE
*/

PARTNOU5 ;
NAMEFinalu5 ;
REV04 ;
DATE21 Gennaio 1989 ;
DESIGNERBrad Fowles ;
COMPANYAnakin ;
ASSEMBLYLucas ;
LOCATION U5 ;
/* PAL16R4B2 */
/* SPECIFICHE DEL PAL */
/* INTERFACCIA PER AMIGA 68020-68881/68000 */
PIN 1 = 7MN ;
PIN 2 = VPA ;
PIN 3 = AS20DLY ;
PIN 4 = QA ;
PIN 5 = QB ;
PIN 6 = QC ;
PIN 7 = QD ;
PIN 8 = BGACK ;
PIN 9 = BG20 ;
PIN 12 = AS00 ;
PIN 13 = HIGHZ ;
PIN 14 = Z2 ;
PIN 15 = BGACK2 ;
PIN 16 = BGACK1 ;
```

```
PIN 17 = VMA ;
PIN 18 = AS20 ;
PIN 19 = BG00 ;

HIGHZ = (BGACK2 ) & (!AS20DLY )#
        (BGACK2 ) & ( BG20 );

!BG00 = (!BG20 )& (!Z2 )& (AS20 );

!BGACK1.D = !BGACK ;

!BGACK2.D = !BGACK1 ;

!Z2.D = (!AS00 )#
        ( AS20 );

!VMA.D = (!QD )  & (!QC ) & (QB ) & (QA ) & (!VPA )#
         (!VMA ) & ( QD )#
         (!VMA ) & ( QC );
/*
DESCRIZIONE: ARBITRAZIONE DEL BUS E GENERAZIONE VMA
*/
PARTNOU6 ;
NAMEFinalu6 ;
REV03 ;
DATE21 Gennaio 1989 ;
DESIGNERBrad Fowles ;
COMPANYAnakin ;
ASSEMBLYLucas ;
LOCATION U6 ;
/* PAL16R4B2 */
/* SPECIFICHE DEL PAL */
/* INTERFACCIA PER AMIGA 68020-68881/68000 */
PIN 1 = 16M ;
PIN 2 = AS20DLY ;
PIN 3 = Z3 ;
PIN 4 = VMA ;
PIN 5 = 7M ;
PIN 6 = DS20DLY ;
PIN 7 = 7MB2 ;
PIN 8 = DTACK ;
PIN 9 = QUAL ;
PIN 12 = DTPRELIM ;
PIN 13 = BUFOUT ;
PIN 14 = DTQUAL ;
PIN 15 = DTQUAL1 ;
PIN 16 = Z1 ;
PIN 18 = DTTRIG ;
PIN 19 = SYSDSPRE1 ;

!DTPRELIM = (!7M ) & ( !7MB2 ) ;

!DTTRIG = (!BUFOUT ) & (!DTPRELIM) ;

!SYSDSPRE1 = !Z1 #
      (!DTACK ) & (!AS20DLY ) & (!DTQUAL) & (!QUAL);

!Z1.D = (!DS20DLY ) & (!Z1 ) #
        (!DS20DLY ) & ( Z3 ) & (!VMA );

!DTQUAL.D = !AS20DLY ;

!DTQUAL1.D = !DTQUAL ;

!BUFOUT = !SYSDSPRE1 ;
/*
DESCRIZIONE: GENERAZIONE DSACK1
*/
PARTNOU7 ;
NAMEFinalu7 ;
REV05 ;
DATE21 Gennaio 1989 ;
DESIGNERBrad Fowles ;
COMPANYAnakin ;
ASSEMBLYLucas ;
LOCATION U7 ;
/* PAL16R4B2 */
/* SPECIFICHE DEL PAL */
/* INTERFACCIA PER AMIGA 68020-68881/68000 */
PIN 1 = HIGHZ ;
PIN 2 = DS20DLY ;
PIN 3 = A0 ;
PIN 4 = SIZ0 ;
PIN 5 = SIZ1 ;
PIN 6 = AS20DLY ;
PIN 7 = CPCS ;
PIN 8 = CPDSACK0 ;
PIN 9 = SRDSACK0 ;
@PROGRAMMI = PIN 11 = SYSDSACK1 ;
PIN 12 = DSACK0 ;
PIN 13 = SRDSACK1 ;
PIN 14 = DSACK1 ;
PIN 15 = CPDSACK1 ;
PIN 16 = AS00BUF ;
PIN 17 = AS00 ;
PIN 18 = LDS ;
PIN 19 = UDS ;

AS00.OE = HIGHZ ;
!AS00 = (CPCS )& (!AS20DLY );

AS00BUF.OE = HIGHZ ;
!AS00BUF = (CPCS )& (!AS20DLY );

UDS.OE = HIGHZ ;
!UDS = (!DS20DLY )& (!A0 ) & (CPCS) ;

LDS.OE = HIGHZ ;
!LDS = ( !DS20DLY ) & ( SIZ1 ) & (CPCS) #
       ( !DS20DLY ) & (!SIZ0 ) & (CPCS) #
       ( !DS20DLY ) & ( A0 ) & (CPCS) ;

!DSACK1 = (!SRDSACK1 )& (!AS20DLY )#
          (!CPDSACK1 )& (!AS20DLY )#
          (!SYSDSACK1)& (!AS20DLY ) ;

!DSACK0 = (!SRDSACK0 )& (!AS20DLY )#
          (!CPDSACK0 )& (!AS20DLY ) ;
/*
DESCRIZIONE: ADDRESS STROBE, DATA STROBE SUPERIORE E
             INFERIORE, GENERAZIONE FINALE DI DSACKX
*/
```

Breakpoint

Terza parte della serie di articoli sul debugging

di Vic Wagner

Vic Wagner iniziò ad avere a che fare con i calcolatori nel 1965 quando divenne parte di un gruppo di studio della US Air Force sulla simulazione digitale del volo. Tornato un civile nel 1966, ha lavorato in quell'epoca, principalmente con costruttori di minicomputer, nell'area del software per sistemi in tempo reale. Nei giorni feriali dalle 8 alle 5, Vic cura la consulenza tecnica telefonica per la Computer Automation Inc. e la manutenzione del software di tre sistemi in tempo reale. Alla sera e nei fine settimana, trascorre il suo tempo conversando con Taarna (il suo Amiga 1000), scrivendo programmi e collegandosi ad AmigaForum. Vic è, inoltre una autorità riconosciuta per quanto riguarda il famoso programma di debugging MetaScope prodotto dalla MetaDigm Inc.

Questo mese esamineremo gli effetti di svariate innovazioni nel campo dei computer e come queste hanno influito sulla verifica dei programmi. Come avremo occasione di vedere, non tutti i cambiamenti sono stati positivi. Anche se molti potrebbero dire che abbiamo fatto passi da gigante per quanto riguarda la scrittura del software e la verifica della sua funzionalità, non sono del tutto sicuro che non si siano fatti anche grandi scivoloni all'indietro.

Diamo un'occhiata alla tecnologia con la quale abbiamo lavorato. Prima vediamo che cosa ha fatto il CRT per noi. Sto parlando del CRT a carattere, più economico di quello grafico. I display CRT esistevano già da qualche tempo, ma il loro costo si manteneva generalmente abbastanza alto. Intorno al 1965 (mi sembra), Raytheon inventò il dispositivo di visualizzazione di caratteri a scansione d'immagine. Questo permetteva l'uso di economici sistemi video per visualizzare i caratteri. I costi dei televisori stavano diminuendo e la scansione d'immagine è proprio il metodo utilizzato dalla televisione. La tecnologia necessaria alla produzione di questo tipo di monitor era ben nota. Per i veri tecnici in ascolto, si trattava di poter utilizzare l'oscillatore orizzontale per aiutare a generare l'alto voltaggio necessario per il CRT, invece di qualche altro metodo.

Ad ogni modo, i CRT si diffusero su larga scala. Erano periferiche molto veloci, in paragone alle vecchie console/stampanti (che chiameremo TTY (perché la TeleType Corporation aveva costruito macchine molto economiche, composte da una tastiera e da una stampante, che erano molto diffuse in quei giorni). La periferica TTY standard stampava circa 10 caratteri al secondo, mentre un display CRT medio produceva un output di circa 1000 caratteri al secondo. Caspita! Possiamo generare un output cento volte più veloce!

Oh-oh, aspettate un momento, noi non possiamo leggere così velocemente.

Va bene, ma possiamo sempre leggere in seguito, a nostra discrezione!

Oh-oh, ma non c'è carta in questa cosa, dove sono finiti tutti i dati cha abbiamo prodotto? Eravamo abituati a raccogliere la carta dal pavimento quando avevamo stampato una serie di dati con il debugger. Ora non possiamo più.

Il CRT, quindi, portava un beneficio (output più veloce) e qualche svantaggio (solo 24 linee visibili contemporaneamente). Questo causò alcuni miglioramenti nei debugger e sicuramente cambiò il modo di guardare alla memoria. Se non altro, non avevamo mai chiesto di stampare un migliaio di parole sul CRT, quantomeno non intenzionalmente. Quest'ultimo non aveva, inoltre, alcuna capacità di *trattenere* le informazioni come l'onnipresente TTY, così non potevamo andare indietro facilmente per vedere cosa avevamo fatto.

Tempo di cambiare alcune vecchie abitudini. Niente più stampa di qualche centinaio di locazioni alla volta (potevamo tranquillamente andare a bere un caffè mentre veniva stampata tutta *quella* memoria) per segnare le parti importanti e mettere il tutto su una pila di fogli. Mettemmo allora delle limitazioni nei debugger per evitare di spingere le informazioni importanti oltre il *confine*. Se eravamo fortunati, il debugger avrebbe offerto la possibilità di inviare queste parti a una stampante. Se eravamo *molto* fortunati, la stampante si sarebbe trovata nella stessa stanza.

Molto presto avemmo tutti i pezzi di carta possibili e immaginabili sparsi intorno, con parti ripassate con l'evidenziatore. Trovare il pezzo giusto poteva essere un problema. Tempo quindi di rendere i debugger più intelligenti (o, quantomeno, più semplici da usare). Per fare ciò, vennero aggiunti alcuni simboli speciali che l'utente poteva definire a proprio piacimento. Questi simboli erano generalmente del tipo R0-R9, o G0- GF, o simili. A questi potevano essere assegnati degli indi-

rizzi e venivano utilizzati come registri *base* per il debugger. Adesso, per esaminare il contenuto del nostro input buffer (al quale avevamo assegnato il registro base R0) potevamo semplicemente dire *esamina R0* invece di *esamina 123F* (l'indirizzo assoluto del nostro input buffer). Questo permise un notevole risparmio di tempo, più di quanto non possa sembrare a prima vista. Non avevamo più bisogno di avere tutti quei pezzi di carta sparpagliati attorno. Assegnando semplicemente una base ai punti interessanti del programma, potevamo esaminare qualsiasi area desiderassimo. Normalmente è molto più veloce scrivere *LR3* (lista a partire da R3) che non cercare il pezzo di carta relativo.

Venne introdotta nei debugger la capacità di accettare delle espressioni nel loro input. Questo ci permise di scrivere I123+456 per vedere la locazione 579 (la lettera 'I' è l'abbreviazione corrispondente al comando 'Inspect' = ispezionare, esaminare; la useremo negli esempi seguenti). Quando eravamo fortunati, I123R0 veniva tradotto automaticamente in I123+R0. Quando eravamo molto fortunati il debugger aveva simboli speciali per i registri che potevano essere utilizzati nelle espressioni (come, ad esempio, X1, X2 e X3 per i registri di indice). Questo ci permetteva di scrivere IX3 e vedere la memoria alla quale puntava il registro X3. Oppure, se X3 puntava a una struttura della quale ci interessava la quinta word, potevamo scrivere I5X3.

Le cose cominciano veramente ad assumere un aspetto migliore. Adesso, se solo non fossimo costretti a lavorare a queste ore assurde... Questo ci porta al prossimo progresso: i computer time-sharing. Come tutti sappiamo, un computer (generalmente parlando) può fare solo una cosa alla volta. Delle persone veramente straordinarie (non ricordo dove) scoprirono che un computer sprecava gran parte del suo tempo aspettando le periferiche di I/O. Si scoprì che quando l'interfaccia di un editor erano un CRT e un essere umano, moltissimo tempo veniva sprecato attendendo che l'umano inserisse dei caratteri. Un tempo ancora maggiore viene sprecato attendendo che un umano inserisca dei dati in un debugger. Queste persone riuscirono a scrivere dei sistemi che dessero l'illusione a più utenti di avere l'uso esclusivo di una macchina. Potevate proprio sedervi di fronte al vostro CRT e credere di essere i soli a utilizzare la macchina. In effetti, il compilatore e l'assemblatore ci mettevano un po' di più, ma la maggior parte dell'attività consisteva nel pensare, editare o fare il debugging. Inoltre, quando volevate assemblare, potevate sempre andare a prendervi una tazza di caffè mentre la macchina era occupata con il vostro programma.

Adesso potevamo fare il debugging durante le normali ore lavorative. Niente più sessioni di lavoro alle 3 di notte. In pratica non tutto era perfetto; durante il giorno c'erano anche le riunioni. Di conseguenza, molti programmatori scoprirono presto che le *normali ore lavorative* erano le meno produttive e quindi continuarono ad esserci sessioni di programmazione a notte fonda. Imparammo in seguito come lavorare assieme, come suddividere grandi progetti, come condividere dati e file. Imparammo, cosa molto importante, come fare sì che la colpa sembrasse sempre di altre persone, in modo che *noi* non fossimo costretti a fermarci fino a tardi. In realtà questo è ignobilmente falso. In quasi tutti i gruppi nei quali ho lavorato eravamo più interessati a finire il lavoro piuttosto che a determinare di chi fosse la colpa.

Abbiamo tutti sentito il sermone: stavamo per liberarci dei soldi di carta, i fondi sarebbero stati trasferiti elettronicamente. La posta cartacea sarebbe stata una cosa del passato. Tutto sarebbe stato immagazzinato elettronicamente. L'avevamo perfino creduto. Incominciammo a vedere i programmi attraverso una finestra di 24 linee di 80 caratteri. I tempi di compile/link erano diventati tanto corti (minori del tempo impiegato per andare a prendere un listato) che eseguivamo un altro ciclo sul computer piuttosto che camminare fino a dove erano immagazzinati i listati. Adesso non fraintendetemi. Non ho nulla contro l'accorciarsi dei tempi di creazione di un prototipo, ma qui sto dicendo che anche i programmi da commercializzare venivano creati in questa maniera. Ebbene, questo spinse l'arte del debugging nella giusta direzione.

Adesso avevamo debugger che potevano eseguire dei comandi contenuti in un file. Avevamo bisogno di usare dei punti di riferimento simbolici, in quanto non potevamo mai essere sicuri della locazione alla quale il nostro programma sarebbe stato caricato. Non avevamo voglia di fare tutta quella strada fino alla stampante per avere la link map, per poter editare il programma e correggerne gli indirizzi. I computer erano progettati per ridurre il lavoro noioso e per svolgere i compiti ripetitivi. Naturalmente, di tanto in tanto, avevamo bisogno di produrre una quantità notevole di output e la stampante era l'unico strumento adatto, così stampavamo tutto in una volta (in modo da dover camminare fino alla stampante una volta sola). Quindi ci sedevamo per esaminare il listato e l'output del debugger e pensavamo al prossimo attacco.

A questo punto sapete da quali tempi provengo e farei meglio a dirvi che il resto dell'articolo è basato unicamente sulle mie esperienze. Dal 1971 mi sono interessato esclusivamente di mini computer impiegati nel controllo di sistemi real-time. La mia conoscenza delle cose che stanno succedendo nel mondo dei *main-frame* è disgraziatamente incompleta. Sento delle voci che parlano di fantastici debugger per le grandi macchine. Sento parlare dei sistemi di controllo del codice sorgente (Source Code Control Systems) che tengono d'occhio tutto quello che voi e i vostri colleghi fate ai programmi. Sento parlare di gruppi di programmazione composti da centinaia (beh, perlomeno da decine e decine) di programmatori. Il gruppo più grosso nel quale abbia mai preso parte era composto da 24 persone, inclusi i direttori e le segretarie. Le mie idee su come si siano evolute le tecniche di prova dei programmi sono necessariamente influenzate dal provincialismo che ha caratterizzato gli ultimi diciassette (ma ne sono passati veramente così tanti?) anni del mio lavoro. Ho passato questo tempo in due sole compagnie, di cui gran parte nella seconda. Va bene, torniamo adesso al 1969 e al mondo dei mini-computer.

Una delle cose più interessanti nel fatto di lavorare per una compagnia di mini-computer consisteva nel fatto che, per la prima volta nella mia carriera, c'era più di un computer. Dopo tutto, noi li fabbricavamo e li vendevamo, quindi ce n'era sempre qualcuno in giro. In realtà disponevamo di un ben attrezzato laboratorio di sviluppo. Eravamo in quattro a occuparci del software, compreso il direttore, e avevamo quattro computer

(no, non uno per ciascuno; questa compagnia produceva due famiglie di computer, quindi ne avevamo due di ciascun tipo). Avevamo punzonatrici, lettori a schede, TTY, perforatrici e lettori a nastro di carta e perfino una stampante (che veniva condivisa fra le quattro macchine). Non c'era praticamente alcun software per queste macchine e noi stavamo sviluppando il sistema operativo. Il mio compito consisteva nello scrivere l'assemblatore per la macchina a 8 bit.

Avete mai pensato a come si fa a scrivere un assembler per una macchina che non ce l'ha? In effetti, io non ho dovuto affrontare questo problema. Si trattava solo del fatto che la nuova macchina aveva molte caratteristiche nuove e loro volevano che l'assembler fosse totalmente rifatto per sfruttare l'ultimo modello della famiglia. Questo significava che dovevo scrivere il programma due volte. La prima volta utilizzando il vecchio assembler, per ottenere un nucleo essenziale funzionante, quindi ricodificando utilizzando il nuovo assembler e poi partire. Questa è stata un'esperienza interessante. Abbiamo dovuto anche scrivere un nuovo debugger perché eravamo nella fase di cambiamento fra la rappresentazione ottale e quella esadecimale. È stata una cosa che ha creato una grande confusione. Avevamo quasi rinunciato a sperare di potere fare le cose in maniera efficiente (avevamo un sistema a nastro magnetico da 12.5 ips) quando una nuova soluzione *economica* si presentò alle porte. Ebbe inizio una nuova era nella ricerca di problemi/soluzioni.

Dischi e sistemi operativi basati su disco... Una cosa è certa, l'avvento di disk drive economici ha cambiato il mondo dell'elaborazione dati. Adesso ognuno poteva permettersi di avere un pacco di dischi per conto proprio. Nei primi anni ottanta, un pacco da 5 megabyte di dischi removibili da 14 pollici costava all'incirca 100 dollari. Nell'area dei mini-computer (i micro non erano ancora stati inventati) una configurazione molto diffusa consisteva in 5 mega fissi e 5 mega rimovibili. Il prezzo di un sistema completo era sceso fino a eguagliare il salario annuale di un programmatore, così le macchine iniziarono a proliferare. Non che ogni programmatore avesse una macchina (questo avverrà molto più tardi), ma una installazione aveva con buone probabilità più di una macchina a disposizione del personale. I sistemi operativi basati su disco proliferavano e noi potevamo assemblare/compilare i nostri sorgenti a velocità accecanti (in realtà anche la tecnologia dei lettori a schede era diventata alquanto avanzata in quegli anni... Ho lavorato con apparecchiature che potevano leggere 1000 schede al minuto). Potevamo tenere il nostro codice sorgente su dischi portatili di 14 pollici di diametro e spessi 2 pollici. Hmmm, quei dischi non erano però altrettanto facili da editare come lo erano le schede. E' difficile togliere un paio di linee di codice dal disco e rimpiazzarle con altre.

Il fatto che gli esseri umani non possano direttamente manipolare i dati scritti su disco porta alla nascita di un nuovo programma: l'editor. Con questo meraviglioso programma potevamo aggiungere, cancellare e rimpiazzare linee di sorgente nei nostri programmi, proprio come potevamo fare in una pila di schede. Naturalmente i primi editor richiedevano che noi ordinassimo i cambiamenti così che potessero essere fatti in una sola passata, ma non importava, perlomeno non ci trascinavamo più migliaia di schede in giro.

Ci tenemmo, comunque, le nostre schede perforate. Non eravamo abbastanza ricchi da permettere a qualcuno di usare un computer solo per inserire un programma, o, detto in altre parole, di sprecare un intero computer semplicemente per editare un file sorgente. Così inserivamo nelle schede le istruzioni dell'editor per modificare il nostro file sorgente. Portavamo queste schede, insieme ad altre schede di controllo, alla macchina. Lanciavamo l'editor. Lanciavamo l'assemblatore/compilatore. Facevamo eseguire il link. Prendevamo il programma eseguibile scritto su nastro di carta (si, nastro di carta; la perforazione di schede era molto costosa) e ci dirigevamo verso una delle macchine per il debugging. Ah si, non bisogna dimenticare di prendere il listato e la link map dalla stampante.

Penso che in questo caso il grande progresso non fosse dovuto tanto ai dischi e ai sistemi operativi basati su disco, quanto alla disponibilità di più di una macchina. Potevamo permetterci di avere una macchina per lo sviluppo e una (o più) per il debugging. Dovevamo ancora suddividere il tempo di utilizzo fra più programmatori, ma questa suddivisione era molto più ragionevole (non tutte le sessioni di debugging venivano fatte alle 3 di notte). Se eravate abbastanza attenti potevate addirittura portare a termine più cicli di edit/compile/debug al giorno.

Quando i prezzi dei dischi scesero ulteriormente si generò un problema interessante. Divenne molto facile che ogni programmatore possedesse un disco. Questo causò, fra le altre cose, l'esistenza di duplicati di acuni file. Si rivelò molto difficile, nel migliore dei casi, assicurarsi che ogni programmatore avesse l'ultima versione dei file di sistema. Una delle cose più frustranti consiste nello spendere un considerevole quantitativo di tempo nel cercare di rintracciare un problema che è stato già risolto. Questo problema delle copie multiple di un file non verrà risolto fino a quando non arriveremo alle LAN (Local Area Networks), argomento che esula dallo scopo di questo articolo (principalmente perché la mia esperienza con il debugging su queste ultime è limitata a un solo progetto). Per continuare il discorso, successero molte cose quasi nello stesso momento:

- I costi delle macchine diminuirono così che potemmo permetterci di avere macchine sulle quali la gente poteva editare i propri sorgenti (per coincidenza, i lettori di schede perforate sembrarono diventare fuori moda nel mondo dei mini).
- Gli editor diventarono più sofisticati. Potevamo addirittura esaminare una *pagina* alla volta del nostro file e fare modifiche in qualsiasi ordine.
- I linguaggi ad alto livello cominciarono a diventare popolari; in particolare i linguaggi strutturati a blocchi come Algol, Pascal, BCPL e C.

Credo che il mondo della programmazione cambiò radicalmente durante questo periodo. Penso che da società di pensatori e pianificatori che eravamo, ci trasformammo in una società di gente che fa. Questo cambiamento potrebbe anche essere avvenuto nella società americana in generale, ma mi sembrò essere molto più accentuato nell'industria del computer. A mio parere il prossimo passo esasperò la situazione.

I personal computer cominciarono a essere portati a conoscenza di un mondo alquanto impreparato. Naturalmente eravamo

tutti a conoscenza dei microcomputer ma, a meno che non ne stessimo costruendo uno a casa, li guardavamo come giocattoli curiosi. Erano lì per giocarci. Avevano solo 16K di memoria. Non avevano software. Non potevamo nemmeno vedere quello che stavano facendo attraverso un pannello di controllo. Perché avremmo dovuto volerne uno? Potevamo disporre di una memoria (e potenza) dieci volte superiore in *una* delle macchine lì nell'ufficio. Beh, forse se proprio avessimo voluto imparare alcuni fondamenti dell'hardware, ma non si poteva scrivere del vero software su di essi.

Intanto si stavano preparando dei piani insidiosi. Degli individui bene intenzionati stavano cercando di mettere insieme un sistema operativo per questi micro. Si scrivevano assemblatori che giravano su questi computer. Queste macchine erano troppo piccole ed economiche per essere dotate di costose periferiche come i lettori e perforatori di schede e le stampanti, così si dovette trovare un altro sistema. Questo qualcos'altro era il floppy disk. I dischi non avevano una grande capacità (128K), ma era sufficiente. Venne creato il CP/M. Era una meraviglia per il suo tempo. Poteva gestire un file system e possedeva un meccanismo per fare girare i programmi.

Editor, assemblatori e addirittura qualche compilatore vennero sviluppati rapidamente, così come vennero sviluppati alcuni programmi per utenti che non erano nemmeno programmatori (mi viene in mente WordStar). Svariate compagnie producevano micro e ne scrivevano i sistemi operativi, per poi venderli agli uomini d'affari. Queste macchine erano economiche sotto ogni punto di vista. I piccoli uomini d'affari se le potevano permettere. Sempre più gente veniva coinvolta nel business dei microcomputer.

E si trattava proprio di business. Sicuro, queste macchine erano di proprietà di singoli, ma questi erano *hacker* (allora era un termine onorevole), fanatici dell'hardware e di piccole neonate compagnie che tentavano di costruire dei sistemi professionali. L'unico programma di debugging degno di nota (beh, che io ho notato), che emerse da tutto questo fervere di attività, era un debugger della Digital Research dal nome ingegnoso di DDT (che loro dicevano essere l'acronimo di Dynamic Debugging Tool). I microprocessori Intel 8080 e Zilog Z80 stavano cominciando a essere venduti in grandi quantitativi. Naturalmente la maggior parte di essi erano contenuti da qualche parte in scatole che controllavano automatismmi di svariato genere, ma erano comunque presenti e attivi.

Poi scoppiò la bomba: Adam Osborne annunciò una macchina che era portatile (beh, 12 chili). Era fornita di *due* disk drive (100K ciascuno) e di un monitor (con schermo da 5 pollici). Veniva venduta a 2500 dollari circa. Compreso in questo sistema c'era del software del valore di 400 dollari circa, inclusi WordStar (un word processor) e SuperCalc (uno spreadsheet). Era veramente una macchina a basso costo, aveva già del software con cui essere utilizzata e poteva essere utilizzata da gente comune. Non bisognava essere un *tecnico* per poterla usare. SuperCalc e WordStar rendevano la macchina utile alla gente normale. Aveva in dotazione perfino CBASIC e Microsoft BASIC per coloro che volessero scrivere i propri programmi. Osborne I fu venduto in gran numero e furono scritti anche molti programmi che giravano su tutte le macchine CP/M. Il personal computer era arrivato. Persino mia moglie e io ne comprammo uno (anche se solo dopo aver trovato un numero sufficiente di scuse e pretesti che ne giustificassero l'acquisto).

Poi scoppiò l'altra bomba: la International Business Machines annunciò il PC. L'IBM?? Vuoi dire quelli che fanno i *veri* computer? Sì, proprio quelli. La maggior parte dei grandi saccenti del giorno (me incluso) predissero che questo era un *grande* errore da parte dell'IBM. Che cosa ne sapevano in fatto di vendita? Quando mai erano stati capaci di trattare con i clienti senza quei sorrisetti di condiscendenza? La gente normale non avrebbe tollerato quel tipo di atteggiamento. Ebbene, tutti quei sapientoni si sbagliavano. Il lavoro ferveva intorno al PC. Tutti quei manager che avevano avuto a che fare con i dipartimenti MIS (Management Information Systems) si stavano comperando le proprie macchine per fare *word processing* e *data analysis*. Il business dei computer e lo scrivere software gli ricordavano gli esperimenti dell'atollo di Bikini nei primi anni sessanta. Tanto, tantissimo software veniva scritto e (quasi difficile da credere) verificato.

I debugger rimasero largamente immutati, ma apparve una nuova sfaccettatura. I disassemblatori venivano adesso incorporati nei debugger. Potevamo chiedere di vedere la memoria sia come dati (come era sempre avvenuto in precedenza), sia come istruzioni. Questo era di particolare importanza su alcuni microcomputer in quanto era molto difficile decifrare i codici a mano. Adesso potevamo vedere direttamente le istruzioni che il computer stava per eseguire. Non dovevamo neanche procurarci un listato del programma (se questo era scritto in assembler). Dovevamo comunque annotare tutti i cambiamenti che facevamo, per poterli inserire in seguito nel listato, ma eravamo proprio sulla giusta strada. Se solo avessimo potuto evitare di annotare i cambiamenti...

Adesso arriviamo ad Amiga. Il primo personal computer multitasking (direttamente fuori dalla scatola) che potete comperare. Sono sicuro che tutti abbiamo quantomeno giocato con il multitasking di Amiga. Sapete quanto sia facile spingere la finestra del programma corrente sul fondo, magari aprire un altro CLI o aprire la piccola finestra di Uedit ed editare del codice sorgente senza uscire dal programma che stavamo usando. Se siete fatti come me (io sono sempre convinto che il bug che ho appena trovato sia solo l'ultimo), comincerete il ciclo di compile/assemble/link non appena avete terminato la modifica al sorgente. In questo modo, quando avete finito di verificare che il programma sia effettivamente corretto, il prodotto finito sarà già disponibile. Ragazzi, questo processo di verifica/correzione non potrebbe essere migliore... eccetto che...

Vorrei che l'output del programma non distruggesse le informazioni di debugging.

AmigaDOS e Intuition vengono di nuovo in nostro soccorso! Se il debugger apre la sua finestra, l'output del programma apparirà in un posto e le informazioni di debugging in un altro! Che bello! Quando il mio programma esegue le sue istruzioni posso continuare a vedere le informazioni di debugging. Se il debugger può aprire *più* di una finestra, posso vedere i registri in una, un buffer o due in un'altra, qualche variabile in un'altra ancora. Caspita! Il paradiso!!

SoftMail

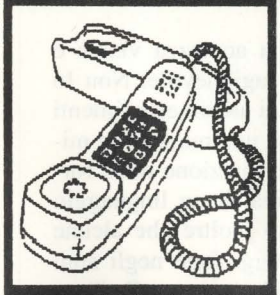
Vendita per corrispondenza di programmi originali ed accessori per computers.

VIA NAPOLEONA 16 - 22100 COMO - TEL. (031) 30.01.74

® SoftMail è un marchio registrato da Lago snc

Presentiamo in questa pagina alcune tra le ultime novità disponibili dal catalogo SoftMail. Ecco alcune informazioni utili per utilizzare il servizio SoftMail: è possibile effettuare ordini telefonicamente SOLO se è già stata effettuata una spedizione a proprio nome ed è stata regolarmente ritirata. Dal secondo in poi accettiamo ordini telefonici. Chi desidera notizie sulla disponibilità ed i prezzi dei prodotti che non compaiono in questa lista può chiamare lo (031)30.01.74 dalle 14:30 alle 18:00. SoftMail può organizzare la consegna anche tramite corriere: interpellateci per maggiori informazioni. Oltre alle ultime novità qui esposte, SoftMail offre l'intero catalogo delle seguenti case: Aegis, Cinemaware, EA, Microprose, Rainbird, SSI, SSG, Sublogic.

Ti è arrivato il catalogo a *colori* SoftMail? Se vuoi riceverlo *gratuitamente*, telefonaci subito!



031/30.01.74

ACCESSORI

Copritastiera A500	25.000
Final cartdrige III	110.000
Flicker master	29.000
Joy. Slik stick	16.500
Joy. SpeedKing	29.000
Joy. SpeedKing Autof.	33.000
Joy. Tac 2	29.000
Joy. Tac 5	39.000
MouseMat tappetino	22.500
Coprimouse	20.000
Portamouse	12.500
Portadischi 3" (30)	34.000
Portadischi 5" (40)	37.000
SlimLine (tastiera 64)	49.000
Voice messenger C64	60.000

LIBRI/HINTS & TIPS

Alternate city clue specificare 8/16 bit	18.000
Bard's tale I clue	22.500
Bard's tale II clue	25.000
Bard's tale III clue	25.000
Black cauldron clue	18.000
Deathlord clue	telef.
Dungeon master clue	25.000
Elite clue	18.000
Graphic adv.creator hint	7.500
Mars saga clue	telef.
Might & Magic clue	25.000
Pool of radiance cle	20.000
Quest for clues	39.000
Sentinel world clue	25.000
Starflight clue	25.000
Ultima V clue	22.500
Wasteland clue	16.500

AMIGA

Heroes lance hints disk	*telef.*
Bard's tale hints disk	*telef.*
Aegis draw 2000	telef.
Aegis images	69.000
Aegis impact	125.000
Aegis sonix	110.000
Aegis videotitler 1.1	185.000
Alternate reality: the city	69.000
Animation multiplane	125.000
Armageddon man	49.000
Audiomaster	85.000
Audiomaster II	telef.
Baal	29.000
Barbarian II	telef.
Batman	29.000
Battlechess	49.000
Black cauldron	9.000
Blitzkrieg at Ardennes	59.000
California games	25.000
Carrier command	49.000
Cell animator	telef.
Chrono quest	65.000
Comic setter	139.000
Comic setter art disks	telef.
Corruption	49.000
Cosmic pirates	telef.
Crazy cars	29.000
Def con 5	59.000
Defender of the crown	59.000
Deluxe print II	99.000
DigiPaint	99.000
DigiView GOLD	telef.
Director	99.000
Disk drive esterno	299.000
Dos2Dos	75.000
Dragon's lair (1 MByte, 6 disks)	75.000
Driller TUTTO italiano	59.000
Dungeon master (1 MB)	59.000
Elite	45.000
Empire	49.000
F16 Falcon	59.000
Fantavision	125.000
Fire & forget	39.000
Fish	45.000
Flight simulator II	99.000
Scenery disks	telef.
Galileo 2.0	99.000
Garfield	49.000
Gauntlet II	25.000
Grabbit	49.000
Hellfire attack	39.000
International soccer	39.000
Italy 90 soccer	39.000
Incr.s.sphere	49.000
Jet & Japan bundle	110.000
Joan of Arc	25.000
King of Chicago	49.000
LED storm	25.000
Legend of the sword	45.000
Life cycles vol.1	39.000
Light!Camera!Action!	99.000
Lords of the Rising Sun	telef.
Major motion	39.000
Maxiplan plus	299.000
Menace	39.000
Mickey Mouse	25.000
Modeler 3D	129.000
Murder on Atlantic	59.000
Nigel Mansell grand prix	49.000
Obliterator	45.000
Offshore warrior	39.000
Outrun	25.000
P.O.W.	59.000
PacMania	25.000
Phantom fighter	45.000
Pioneer plague	59.000
President is missing	49.000
ProSound designer	79.000
ProSound w/hardware	199.000
Prowrite 2.0	179.000
Publisher plus	125.000
Reach for the stars	59.000
Rebel...Chickamauga	telef.
Return to Genesis	39.000
Rocket Ranger	59.000
Roger Rabbit	69.000
SEUCK	telef.
Sex vixens in space	55.000
Sculpt/Animate 4D	telef.
Sentinel	49.000
Sinbad	59.000
Skyfox II	59.000
Starfleet	59.000
Starglider II	49.000
Superman	39.000
Sword of sodan	69.000
The bard's tale II	49.000
The works!	249.000
Three Stooges	59.000
Tracker	49.000
TV Sports: Football	59.000
Ultima IV	49.000
Universal military sim.	45.000
Data disks 1 e 2	telef.
Victory road	39.000
Videoscape 3D 2.0	250.000
Designs disks	telef.
VIP professional	199.000
Virus	29.000
Virus infection protection	89.000
WB Extras	69.000
Whirligig	39.000
Willow	59.000
World Cl. Leaderboard	25.000
Zak McKracken	59.000
Zing keys	25.000
Zoetrope	199.000
Zoom	45.000
3 Demon	145.000

COMMODORE 64 CASS.

After burner	18.000
Barbarian II	18.000
Batman	18.000
Dragon ninja	15.000
Exploding fist+	15.000
G.I.hero	15.000
Italy 90 soccer	18.000
Last Ninja II	25.000
MicroSoccer	telef.
Robocop	18.000
R-type	18.000
Serve & volley	22.000
Shoot'em up constr. kit	25.000
Stunt bike simulator	5.000
Tiger road	15.000
Total eclipse (ital.)	15.000
4 soccer simulator	18.000

COMMODORE 64 DISCO

ADD: Heroes of the lance	telef.
ADD: Pool of radiance	59.000
American civil war III	49.000
Barbarian II	25.000
Deathlord	35.000
Driller	29.000
Echelon (con LipStick)	59.000
Eternal dagger	35.000
Fast break	29.000
Fish	29.000
Game over I+II	25.000
GeoPublisher	100.000
Grand Prix Circuit	telef.
Gulf strike	49.000
Home video producer	69.000
Italy 90 soccer	25.000
LED storm	15.000
Mars saga	35.000
McArhtu's war	49.000
MicroSoccer	telef.
Neuromancer	39.000
One on one II	29.000
Powerplay hockey	29.000
Red storm rising	49.000
Rocket Ranger	telef.
Shoot'em up constr. kit	35.000
Stealth mission	75.000
T.K.O.	29.000
The bard's tale III	35.000
Total eclipse (ital.)	20.000
Ultima V	49.000
Wasteland	39.000
Wec Les Mans	21.000
Who framed Roger Rabbit	telef.
Zak Mc Kracken	39.000
4 soccer simulator	25.000

COMMODORE 128 DISCO (128K, 80 COL.)

"C" Language	99.000
Basic 8.0	75.000
GEOS 128	telef.
Mach 128	125.000

OFFERTE SPECIALI

(FINO AD ESAURIMENTO)

		Valore	A sole
FIREBIRD SPECIAL			
AMIGA	Bubble Bobble+Whirligig+ Enlightenment	107.000	75.000
C64 cass.	Savage+Intensity+Sentinel	48.000	33.000
C64 disco	Savage+Intensity+Sentinel	65.000	39.000
AEGIS ENTERTAINMENT			
AMIGA	Arazok's Tomb+Ports of Call	120.000	89.000

DEFENDER OF THE CROWN C64: cass. 12.000 disco 15.000

... SOFTMAIL TI DA' DI PIU'.

BUONO D'ORDINE da inviare a: **LAGO DIVISIONE SOFTMAIL**, VIA NAPOLEONA 16, 22100 COMO, FAX (031) 30.02.14, TEL (031) 30.01.74

Desidero ricevere i seguenti articoli:

☐ Addebitate l'importo sulla mia CARTASI/MASTERCARD/VISA/AMERICAN EXPRESS nr. _____ scadenza _____

☐ Pagherò al postino in contrassegno

Titolo del programma	Computer	Cassetta/disco/accessorio	Prezzo

Contributo spese di spedizione Lit. 5.000

TOTALE LIT. _____

ORDINE MINIMO LIT. 20.000 (SPESE ESCLUSE)

Cognome e nome _____

Indirizzo _____ Nr. _____

CAP _____ Città _____ Prov. _____ Telefono _____

FIRMA (SE MINORENNE QUELLA DI UN GENITORE)

VERRANNO EVASI SOLO GLI ORDINI FIRMATI _____

TA003

Programmare in Modula-2 con Amiga

Acutezza visiva: buona tecnica, grande confusione

di Clark Williams

Clark ha programmato per 19 anni in ogni linguaggio a partire dal COBOL e RPG, fino all'assembly dal 68000, l'IBM OS360, DEC e persino sistemi PICK. Ha parlato 123 differenti linguaggi di programmazione (senza contare i dialetti del Pascal, Fortrash e così via) ma il suo primo amore è il volo. Quando non è in volo, scrive programmi in Modula-2.

Questa sarà una collezione di argomenti che trattano della programmazione di Amiga utilizzando il linguaggio Modula-2. Discuteremo svariati argomenti specifici di Amiga, così come alcuni aspetti generali della programmazione: standard per la codifica, filosofia di progettazione, acutezza visiva.

Indubbiamente queste discussioni saranno di natura religiosa. Molti programmatori credono molto fortemente negli standard per la codifica.

Molte opinioni sugli standard di codifica sono fortemente imposte negli anni in cui i programmatori vengono formati; ciò avviene frequentando classi nelle quali i docenti posseggono forti ed incrollabili opinioni che sono destinate ad essere tramandate agli studenti. Queste opinioni sono spesso tramandate senza che l'istruttore senta alcuna necessità particolare o vincolante, di sottolineare che sono state ottenute da questo insegnante dal suo maestro preferito mentre stava insegnando in qualche sala piena di fertilizzante (Humor americano !! NdT).

Le opinioni che leggerete sono personali e basate su esperienze di lavoro e di studio. Ve le offro non per convincervi che io abbia ragione, o che voi abbiate torto, ma per darvi la possibilità di seguire i miei esempi di codice e/o i frammenti di procedure con estrema semplicità.

Dovreste provare gli esempi che sono l'essenza di questi articoli. Anche se conoscete già il Modula-2, dovreste provare lo stesso gli esempi e prammaticamente giocare con loro per solidificare la vostra conoscenza di Amiga.

Gli standard di codifica, quando esistono, sono una delle opinioni più fermamente tenute dai programmatori. Ogni programmatore sembra possedere il suo insieme di standard. Ogni simbolo di punteggiatura di questi standard personalmente tenuti viene difeso con sforzi sovrumani. Spesso tale difesa viene espressa in un linguaggio pieno di metafore grafiche i cui colori ed immagini farebbero sfigurare Amiga. Senza dubbio i miei standard sembreranno stupidi o intelligenti a seconda che voi concordiate o meno con i miei standard personali. Comunque, gli standard sono importanti, anche critici, per una presentazione adeguatamente costruita delle tecniche di programmazione.

I miei standard sono basati su di un concetto chiamato "acutezza visiva". Se uno degli scopi della programmazione strutturata è produrre programmi che siano leggibili sia dalle macchine che dalle persone, allora dobbiamo fare alcune considerazioni su quello che significa "forma leggibile" nella presentazione dei programmi ai programmatori. La presentazione del linguaggio al programmatore dovrebbe consentire una rapida individuazione degli elementi del linguaggio. Una rapida occhiata dovrebbe costituire tutto il necessario per individuare una condizione o una istruzione, ecc.

La acutezza visiva è una misura della rapidità con la quale una persona estranea al codice ma dotata di familiarità con la sintassi di un linguaggio può trovare un elemento di quel linguaggio e comprendere il contesto di quell'elemento.

Molte persone mi hanno suggerito che la acutezza visiva è troppo soggettiva per essere una misura ragionevole. Non lo condivido! Basato sull'evidenza empirica di alcuni esperimenti informali, penso che ogni linguaggio abbia una massima e misurabile acutezza visiva. Penso che la presentazione della forma che porta alla massima acutezza visiva di un linguaggio possa essere misurata e dimostrata. Penso inoltre che alcune delle discussioni riguardanti alcuni linguaggi fatte negli anni passati siano state fatte senza alcuna considerazione sulla "leggibilità", cioè sulla acutezza visiva.

Credo inoltre che i linguaggi possono e dovrebbero essere confrontati (all'interno del contesto del compito da implementare). L'acutezza visiva dovrebbe essere una parte importante di questa comparazione.

Vi sono molti fattori da considerare quando si scelgono presentazioni visivamente acute, alcuni di questi sono:

- Presentazioni su stampante piuttosto che su monitor,

1) Condizioni del nastro, inchiostro, ecc.

- Sintassi e costruzione del linguaggio,

  1) Istruzioni IF-THEN-ELSE.

  2) Spazi bianchi.

  3) Commenti, utilizzo e contenuto.

- Come è semplice usare gli elementi di un linguaggio allo scopo di aumentare la acutezza visiva di quel linguaggio.

  1) Identificatori-nomi (ad esempio l'utilizzo del carattere '_')

Quali sono gli scopi nell'aumentare la acutezza visiva?

- Ridurre i tempi di correzione.

- Ridurre i tempi di apprendimento per i nuovi utenti del linguaggio.

- Rendere le presentazioni standard, aumentando perciò le comunicazioni circa il linguaggio, includendo le sue strutture, come programmi, moduli, routine, elementi, pacchetti, etc.

Se avete letto qualche libro sulla programmazione strutturata, avrete notato le parole "l'uso appropriato dello spazio bianco (linee vuote) e dei commenti è utile nel rendere il vostro codice leggibile." Quello è solitamente l'ultimo commento che si può leggere, proveniente dall'autore che vi assiste nel rendere i vostri programmi più leggibili.

Cominciamo con una discussione sugli identificatori. Gli identificatori dovrebbero essere semplici da leggere. Alcuni linguaggi utilizzano il carattere '_' per spezzare un identificatore. Così un identificatore come:

```
buffers richiesti per il task switching
```

apparirebbe come:

```
buffers_richiesti_per_il_task_switching
```

Se tentate di introdurre questo identificatore scoprirete che il carattere '_' è posizionato in un posto molto sconveniente sulla tastiera. Potreste anche concordare con il fatto che con povere condizioni di nastro il carattere '_' può scomparire e l'identificatore può essere confuso con identificatori simili.

Nel raggiungere una alta "acutezza visiva" dobbiamo considerare svariate condizioni sui formati della presentazione. Comunque, questo non è un articolo sulla "acutezza visiva" così non analizzeremo queste condizioni. Invece, ho scelto un formato che risolve molti dei problemi presentati da quelle condizioni che si incontrano in una tipica struttura di programmazione.

Alcuni di questi standard sono "regole". Alcuni di questi standard sono "consigli". Le regole sono intese da seguire il più rigorosamente possibile. Possono anche essere infrante ma non è questo lo scopo. I consigli sono proprio quello. Vi sono d'aiuto quando vi stufate e non avete tempo per essere creativi. Ciascun standard viene descritto più avanti come una regola o un consiglio. Non tutti gli standard sono descritti in questo articolo ma via via che si incontreranno negli esempi verranno classificati.

Regola #0: Gli identificatori dovrebbero possedere la prima lettera di ogni parola/abbreviazione/sigla mnemonica maiuscola e solo la prima lettera. Ad esempio, il nostro identificatore di prima apparirebbe come:

```
BuffersRichiestiPerIlTaskSwitching
```

Questo significa che gli identificatori contenenti parole come USA verrebbero scritti come:

```
UsaKeyboardSwitching.
```

Non è semplice come il testo pulito, ma richiede un piccolo sforzo per abituarsi ad esso - provatelo!

Consiglio #0: Il ";" e il suo utilizzo è un altro argomento caldamente discusso. Io farò uso del ";" così come il Dr. Wirth (il papà del Pascal e del Modula-2) intendeva che fosse utilizzato in Modula-2. Il ";" verrà utilizzato come un separatore di comandi, non come un terminatore di comandi. Perciò, non vedrete un ";" prima di una istruzione END o prima di un UNTIL, etc.

Consiglio #1: I commenti sono un elemento di molti linguaggi che sembrano essere un qualcosa di successivo nel disegno del linguaggio.

Alcuni linguaggi sono peggiori da commentare rispetto ad altri; comunque, in Pascal e in Modula-2 accade di avere commenti che possono essere facilmente ed utilmente impiegati nel codice.

Come metodo personale di commento e di utilizzo del ";" tenderò a raggruppare i commenti utilizzando il ";". Se un commento appartiene a una particolare istruzione sarete in grado di stabilire a quale istruzione appartiene con l'uso del ";": sarà posizionato alla fine del commento per i commenti che seguono l'istruzione cui si riferiscono, e alla fine della istruzione per quelli che precedono l'istruzione in oggetto.

Pertanto il commento nel seguente frammento di codice:

```
Fragmentation := TRUE;

(* Informa la routine chiamante dell'errore *)

Error := TRUE;
```

è in riferimento alla istruzione "Error := TRUE" in quanto giace fra l'istruzione precedente e il ";". Comunque, se avessimo scritto il frammento di codice nel formato seguente:

```
Fragmentation := TRUE
(* Informa la routine dell'errore *) ;
```

```
  Error := TRUE;
```

il ";" associa il commento alla prima istruzione nel frammento di codice: "Fragmentation := TRUE".

Ancora, questo è un utilizzo personale del ";" e dei commenti. Costituisce qualcosa che rende il codice più semplice da leggere per me (e per gli altri). Se non avete intenzione di utilizzarlo come una convenzione, spetta a voi deciderlo; ma avrete bisogno di conoscerlo per seguire più attentamente il mio codice.

Consiglio #2: Per identificare un segmento di codice, utilizzate il commento all'inizio del segmento di codice ed utilizzate gli spazi per aumentare la acutezza visiva.

Ad esempio:

```
FOR Index := 1 TO EndOfCurrentList DO
  (* cerca tutti gli stop. *)
  ...
END (* cerca tutti gli stop. *)

(*
  Il seguente ciclo REPEAT-UNTIL elaborerà gli
  stop introdotti nella linea di comando. Tutti
  gli stop sono elaborati fino a quando non
  viene incontrato un errore durante
  l'elaborazione. Vedere le specifiche
  funzionali pagina 3-144.
*)
REPEAT (* effettua la ricerca degli stop *)
  ...
  UNTIL (* Non sono stati incontrati stop
  oppure si è verificato un errore che può
  essere determinato da *)
    ( Error ) OR ( AllStop );
```

Consiglio #3: Probabilmente il maggior dibattito nel mondo degli standard di stesura del codice riguarda l'istruzione IF-THEN-ELSE. Il mio scopo principale è quello di rendere l'istruzione e le sue condizioni facili da trovare, e di mostrare chiaramente quali condizioni sono associate a quali istruzioni.

Vi sono due modi di utilizzare l'istruzione IF-THEN-ELSE. Il primo è:

```
IF <condizione booleana> THEN
  <istruzione successiva a then>
ELSE
  <istruzione successiva a else>
```

Il secondo metodo è:

```
IF <condizione booleana>
  THEN (* commento *)
    <istruzione successiva a then>
  ELSE (* commento *)
    <istruzione successiva a else>
```

Io preferisco il secondo metodo per le ragioni seguenti:

1) Questo metodo è, per evidenza empirica, più visivamente acuto.

2) La istruzione è una istruzione IF con due diramazioni associate. Le istruzioni appartenenti alla diramazione THEN dovrebbero ovviamente essere associate a quella condizione. Quando il THEN viene utilizzato come il terminatore sintattico della espressione booleana a volte non sapete da un rapido esame se l'istruzione IF è o non è tutta su una linea.

3) La proprietà delle istruzioni dovrebbe essere ovvia. La istruzione IF possiede le condizioni THEN ed ELSE e il THEN ed ELSE possiedono certe istruzioni. L'indentazione mostrata nel secondo metodo non lascia alcun dubbio su quali istruzioni appartengono a certe condizioni e quali condizioni appartengono a quali istruzioni. Questo rende più facile identificare ciascun segmento di codice e perciò aumenta la 'acutezza visiva'.

L'ambiente di programmazione TDI Modula.

Ora avete una idea di come io organizzo il mio codice. Diamo una occhiata a un po' di codice per Amiga. In particolare, vediamo come iniziare a utilizzare l'implementazione TDI del Modula-2. Cominceremo esaminando l'ambiente disponibile quando iniziamo un programma in Modula-2.

Utilizzeremo l'implementazione del Modula-2 TDI V3.00b. L'indirizzo di TDI è 10410 Markison Road, Dallas, TX 75238, (214) 340-4942.

Questo non sarà un dibattito per approvare e tantomeno per parlare male. So che molti di voi hanno ricevuto un trattamento da TDI che non era molto professionale. Capisco le vostre frustrazioni, ma forse possiamo affrontare il problema attraverso questa trattazione.

Una precisazione sul Modula-2. Ci sono svariati tipi di "moduli" nel Modula-2. Ci sono moduli di programmi, moduli di libreria e moduli "client". Un modulo programma è facilmente identificabile: inizia sempre con il termine MODULE. Un modulo "client" chiama un modulo libreria. Un modulo libreria viene chiamato da un modulo "client".

Ovviamente, questo insieme di moduli libreria "client" dipende da quale punto stiamo esaminando nella esecuzione di un programma. Potreste avere un modulo programma "A", il quale è "client" per un modulo libreria "B", che è "client" per un modulo libreria "C", etc.

Tutti i moduli libreria hanno due parti:

1) Il modulo definizione.

2) Il modulo implementazione.

Regola #1: Dovreste sempre definire prima l'interfaccia a un modulo, programma, pacchetto, libreria o funzione che sia. La implementazione del modulo sarà più semplice quando la definizione viene fissata per prima.

Questo non significa che non cambierete mai la definizione

della interfaccia che state progettando. Potete benissimo gettarla e ricominciare tutto da capo. In ogni caso, dovreste prima pianificare che cosa volete come input e output e poi rivedere, rivedere, rivedere. Scrivete domande e scrivete le risposte a queste domande quando le trovate. Questo genere di cose vi guiderà nella costruzione di interfacce migliori, moduli migliori e codici migliori.

L'Amiga Developer's Manual (Bantam Books) descrive un ambiente per i linguaggi che intendono fare uso dell'AmigaDOS. In particolare descrive come sia gli ambienti C ed assembler hanno le librerie Exec e DOS già aperte per gli utilizzatori del linguaggio. Ci si aspetta ovviamente che tutti i linguaggi aprano queste due librerie.

Inizieremo con il seguente modulo programma per vedere come in effetti le librerie appropriate sono aperte utilizzando il TDI Modula-2.

Scopo: Determinare quali librerie vengono aperte entrando nell'ambiente TDI Modula-2.

Il piano di attacco è di controllare gli indirizzi di base delle librerie alle quali siamo interessati. I due indirizzi base delle librerie che desideriamo sono ExecBase e DOSBase. La definizione per ExecBase è reperibile in un modulo che è molto importante per il TDI Modula-2 chiamato AMIGAX. Discuteremo in seguito di questo modulo. Il secondo indirizzo è reperibile in un modulo chiamato DOSLibrary. Il modulo programma dovrebbe controllare i valori in questi indirizzi come prima istruzione di controllo. Se questi contengono degli zeri allora le librerie non sono aperte. Se non contengono zeri allora le librerie sono già aperte per voi. Dal momento che scrivere sullo schermo potrebbe IMPORTare un modulo che apre la DOSLibrary (ma non la chiude per voi in seguito), non faremo uso dell' I/O di schermo. Al contrario, scriveremo un file su disco dal nome "AnswerFile.dat" che conterrà due linee che ci diranno lo stato della libreria all'inizio del programma.

Questo non sarebbe necessario se conoscessimo lo stato delle librerie dalla documentazione. Ma TDI non ci fornisce alcuna informazione specifica sullo stato del Modula-2 su Amiga quando iniziamo.

Il modulo programma che realizza il nostro scopo viene mostrato nel primo listato ed è chiamato CheckEnvironment.

Quando viene eseguito, un file chiamato "AnswerFile.dat" verrà individuato nel drive interno (df0:). Il file può essere visualizzato sullo schermo tramite il comando TYPE da una finestra CLI. Il contenuto dovrebbe apparire in questo modo:

```
Exec era aperta.
DOS non era aperta.
```

Ovviamente, in ambiente TDI Modula-2 la DOSLibrary non viene aperta automaticamente. Questo è contrario all'ambiente di libreria standard Amiga; in ogni caso, questo ci offre l'opportunità di scrivere un modulo libreria che definirà il nostro ambiente come lo standard mostra che dovrebbe essere definito.

Perché non possiamo semplicemente aprire sempre DOSBase? In effetti potremmo. Ma non facendo uso della istruzione arbitraria:

```
DOSBase := OpenLibrary (DOSName,0)
```

Il motivo è che esiste un modulo che a volte apre la libreria DOS per conto suo. Ma questo modulo non si cura di chiudere la libreria dal momento che non può stabilire quando il programma ha terminato di utilizzare il modulo. Il modulo è MODULE Streams.

Così, per stabilire quando aprire DOS - e quando non aprirlo - dovremmo sempre includere la istruzione:

```
IF DOSBase = SYSTEM.NULL THEN
    DOSBase := Libraries.OpenLibrary(DosName,0)
END;
```

Si capisce che vi sono un paio di moduli dove questo genere di programmazione accidentale viene fatta (vedi in particolare il MODULE Terminal). Sicché, quello che faremo invece sarà costruire un modulo che aprirà le librerie che noi designamo tra DOS ed Exec ed aprirà sempre il DOS se e solamente se non è già stato aperto.

Questo significa che se mai useremo il modulo Terminal che dovrà chiudere la libreria DOS prima della chiamata a Terminal, o in seguito chiuderemo due volte la libreria DOS. Tenete presente che AmigaDOS tiene memoria di quali librerie sono aperte e di quante volte quella libreria è stata aperta.

Queste routine in effetti riserveranno spazio per le librerie di Amiga. Chiameremo il modulo che descrive queste routine AmigaHouseKeeping.

Routine per la manutenzione delle librerie.

Necessitiamo di alcune routine per gestire la apertura e la chiusura delle librerie utilizzando TDI Modula-2 e AmigaDOS. Che cosa faremo? Per cominciare seguiremo alcune delle regole e dei consigli che sono stati definiti. Una delle prime era stabilire che cosa stavamo per fare. Iniziamo pertanto con la progettazione.

Che tipo di funzionalità dovremmo avere nelle nostre routine? Ci piacerebbe essere in grado di chiamare una routine che apre le librerie DOS ed Exec se non erano già state aperte. Ci piacerebbe inoltre una routine che aprisse qualsiasi libreria da noi specificata. Ognuna di queste routine dovrebbe avere anche il logico opposto: routine per chiudere le librerie.

Dal momento che utilizzeremo librerie, vediamo di creare un tipo che sarà rappresentativo delle librerie che adopereremo. Queste saranno librerie che potremo mettere nello scaffale del nostro Workbench. Avremo bisogno di un set che ci dirà quali librerie abbiamo/avevamo aperto.

```
TYPE

LibraryShelf = ( Exec, DiskFont, Dos,
    Graphics,Intuit, Layers, Translate );
```

```
OnTheShelf = SET OF LibraryShelf;
```

Non includiamo la mathlibrary in queste perché, nell'implementazione del TDI Modula-2, la mathlibrary non è un qualche cosa a cui si fa accesso come una libreria. Questa è già disponibile. Non aprite la MathLibraryBase; questa variabile non esiste. Semplicemente utilizzate le funzioni nel momento in cui ne avete bisogno.

Se avessimo un intero scaffale di librerie, avremmo tutte le librerie aperte. Se stiamo facendo cose molto semplici e non specifiche dell'AmigaDOS probabilmente vogliamo aperte soltanto DOS ed Exec. Pertanto...

```
CONST

FullShelf = OnTheShelf { Exec..Translate };

NormalShelf = OnTheShelf { Dos };
```

Notate che non includiamo Exec nello scaffale normale. Ricordate che abbiamo visto in precedenza che la libreria Exec è sempre aperta nel momento in cui iniziamo. Così effettuiamo solo un controllo per il DOS nello scaffale normale.

Ora siamo pronti a discutere una routine. Questa routine ricorderà per noi quali librerie erano aperte quando abbiamo iniziato. Vogliamo che la routine ci restituisca un valore che ci permetta di stabilire quali librerie erano "nel nostro scaffale" nel momento in cui abbiamo chiamato la routine. Questa routine sarà una funzione "procedura".

Piccola parentesi...Credo che uno degli errori più evidenti che Dr. Wirth abbia commesso con il Modula-2 sia questo. Esiste una netta differenza tra PROCEDURE, che non restituiscono valori attraverso i loro identificatori, e FUNCTION che invece lo fanno. Le procedure non possono essere utilizzate nelle espressioni. Le funzioni possono essere utilizzate nelle espressioni, ecc. In Modula-2 esistono Procedure e Funzioni. Le funzioni procedure sono FUNCTION. Speravo che Dr. Wirth lasciasse solo gli identificatori di parole chiave tra Pascal e Modula-2. Rende il codice molto più visivamente acuto. "Diamine, questo non è ciò che è accaduto".

Possiamo allora descrivere gli input e output della nostra funzione. Non necessitiamo di alcun input da parte del programma chiamante. Invece abbiamo proprio bisogno di controllare l'indirizzo "base" delle librerie. Questo richiederà che la funzione sia in grado di accedere ai valori (cioé IMPORTare questi indirizzi di base) degli indirizzi di base delle librerie. Restituiremo un insieme che mostra le librerie aperte. Restituiremo un insieme vuoto se nessuna delle librerie era aperta (un errore dal momento che sarebbe meglio che la Exec fosse aperta). Altrimenti restituiremo un insieme che indica quali librerie erano aperte. Se questo insieme include una libreria, allora questa era aperta; altrimenti, questa era chiusa. Esaminate la funzione chiamata RememberOpenLibraries nel Listato 2 per vedere come è stata effettuata l'implementazione.

La seconda procedura aprirà le librerie che noi abbiamo richiesto. Abbiamo bisogno di essere in grado di specificare quali librerie dovrebbero essere aperte e avremo bisogno di sapere se le librerie sono state aperte con successo. Possiamo specificare le librerie da aprire attraverso una variabile passata come un parametro attuale. Se tutte le librerie sono aperte correttamente otteniamo un valore TRUE di ritorno dalla funzione.

Il problema, come sempre, è quello di gestire un errore. Se le librerie non sono aperte correttamente, vogliamo sapere quali librerie non è stato possibile aprire. Possiamo fare questo facendo sì che il parametro attuale serva per un doppio scopo. In input, esso specifica quali librerie devono essere aperte; in output, indicherà le librerie che non è stato possibile aprire, se è intervenuto un errore, oppure restituirà l'insieme vuoto se non si sono verificati errori. Questo approccio realizzativo richiede che il parametro attuale sia un parametro chiamato per referenza.

La procedura-funzione che esegue tutto questo è chiamata OpenLibraries e il listato corrispondente è il secondo. Notate che abbiamo utilizzato le costruzioni {SET} - {SET} e {SET} + {SET} nelle procedure-funzioni. Avremmo anche potuto utilizzare INCL({SET},elemento) come costrutto. In ogni caso io preferisco la forma predicate calculus di queste equazioni piuttosto che gli esempi di funzioni. Li ritengo più chiari e più visivamente acuti. Potete cambiare questi costrutti in accordo alle vostre preferenze a condizione di mantenere una certa consistenza nel loro utilizzo.

La procedura-funzione CloseLibraries a un esame dei requisiti è la stessa della procedura-funzione OpenLibraries. Richiede gli stessi input e output cosicché la procedura-funzione appare praticamente la stessa.

La prossima procedura-funzione è la routine che necessita di aprire le librerie di Amiga per noi come se stessimo normalmente utilizzando le librerie. Ovvero, dovrebbe sempre assicurarsi che DOS sia aperta, e ogni altra libreria che noi specifichiamo in un parametro attuale passato alla procedura-funzione. La routine di chiusura apparirà praticamente identica; comunque, il valore restituito nel parametro attuale sarà leggermente differente da quello che abbiamo descritto nelle routine precedenti. Il parametro non ci restituirà le librerie che non è stato possibile aprire, ma restituirà le librerie che sono state aperte correttamente sia nel caso di errore o no.

Solitamente non è una buona norma utilizzare due differenti convenzioni di chiamata per delle routine simili; comunque, molte volte questo è quello che deve essere fatto. Il problema nasce esaminando la natura delle routine e dei moduli. I moduli sono una collezione di routine che sono simili nei loro compiti, per esempio un modulo matematico o un modulo grafico. Il mio preferito consiglio realizzativo è utilizzare una interfaccia logica di routine, piuttosto che tentare di essere consistenti in tutte le routine all'interno di un modulo. Nel mio esempio, le interfacce per OpenLibraries e AmigaOpenHouse sono completamente differenti, ma sono nello stesso modulo in quanto sono logicamente raggruppate.

La nostra routine allora è chiamata AmigaOpenHouse. L'idea è quella di aprire ogni cosa che viene specificata. Il codice che esegue questo prende tutto ciò che ritorna RememberOpenLi-

braries (le librerie già aperte), cancella quegli elementi dalla richiesta delle librerie aperte, cancella exec (dovrebbe essere già aperta) e apre il rimanente. Il nostro valore di chiamata viene lasciato indisturbato fino a che non sussiste un errore nell'apertura di una libreria. In questo caso il valore restituito è costituito dalle librerie che sono state aperte e un valore FALSE per la funzione.

L'inizializzazione del modulo non richiede alcun codice.

Questo è il modulo. Il modulo programma nel Listato 3 mostra come utilizzare (chiamare) le routine.

Listato 1: CheckEnvironment

```
MODULE CheckEnvironment;

FROM SYSTEM     IMPORT ADR, NULL;
FROM DOSLibrary IMPORT DOSBase, DOSName;
FROM DOSFiles   IMPORT Open, Close, FileLock,
                       FileHandle, Lock, Unlock,
                       AccessWrite, ModeNewFile,
                       Write;
FROM Libraries  IMPORT OpenLibrary, CloseLibrary;
FROM AMIGAX     IMPORT ExecBase;
FROM Strings    IMPORT Assign, Concat, Length;

CONST
 (* ottale per il line feed ANSI ASCII. *)
 LineFeed       = 12C;
 (* ottale per il carriage return ANSI ASCII. *)
 CarriageReturn = 15C;
 ExecHeader     = "Exec e':";
 DosHeader      = "DOS e':";
 NotHeader      = " NON";
 OpenHeader     = " aperto.";
 AnswerFileName = "df0:AnswerFile.dat";

TYPE
 CharArrayPtr = POINTER TO CharArray;
 CharArray    = ARRAY [ 0..79 ] OF CHAR;

VAR
 ExecAlreadyOpened : BOOLEAN;
 DosAlreadyOpened  : BOOLEAN;
 StringArray       : CharArray;
 CharBuffer        : CharArrayPtr;
 AnswerFileHandle  : FileHandle;
 Error             : LONGINT;

BEGIN (* CheckEnvironment *)
 ExecAlreadyOpened := ExecBase  NULL;
 DosAlreadyOpened  := DOSBase   NULL;

IF NOT DosAlreadyOpened
  THEN DOSBase := OpenLibrary ( DOSName, 0 )
 END;

 AnswerFileHandle := Open ( AnswerFileName,
ModeNewFile );

 IF AnswerFileHandle  FileHandle ( NULL )
  THEN (* Il file e' stato aperto *)
   (*
      Crea la stringa "Exec e': aperto." oppure
      "Exec e': NON aperto."
   *)
   Assign ( StringArray, ExecHeader );
   IF NOT ExecAlreadyOpened
    THEN Concat ( StringArray, NotHeader,
                  StringArray )
   END;

   Concat ( StringArray, OpenHeader, StringArray );

   (*
      Tenete a mente che Length e' un contatore di
      caratteri, ma che gli array iniziano con
      indice a zero.
   *)

   StringArray [ Length ( StringArray ) ] :=
         CarriageReturn;
   StringArray [ Length  (StringArray ) ] :=
         LineFeed;

   (* Imposta la scrittura su Amiga. *)

   CharBuffer := ADR ( StringArray [ 0 ] );
   Error := Write ( AnswerFileHandle,
                    CharBuffer,
                    LONG ( Length ( StringArray ) ) );
   (*
      Crea la stringa "DOS e': aperto." oppure
      "DOS e': NON aperto."
   *)

   Assign ( StringArray, DosHeader );

   IF NOT DosAlreadyOpened
    THEN Concat ( StringArray, NotHeader,
                  StringArray )
   END (* IF *);

   Concat ( StringArray, OpenHeader, StringArray );

   (*
      Tenete a mente che Length e' un contatore di
      caratteri, ma che gli array iniziano con
      indice a zero.
   *)

   StringArray [ Length ( StringArray ) ] :=
         CarriageReturn;
   StringArray [ Length ( StringArray ) ] :=
         LineFeed;

   (* Imposta la scrittura su Amiga *)

   CharBuffer := ADR ( StringArray [ 0 ] );
```

```
   Error := Write ( AnswerFileHandle,
                    CharBuffer,
                    LONG ( Length ( StringArray ) ) );
   Close ( AnswerFileHandle )
 END (* IF *)
END CheckEnvironment.
```

Listato 2: AMIGA Housekeeping, definizione.

```
DEFINITION MODULE AmigaHousekeeping;

FROM SYSTEM             IMPORT NULL;
FROM Libraries          IMPORT CloseLibrary,
                               OpenLibrary;
FROM AMIGAX             IMPORT ExecBase;
FROM DiskFontLibrary    IMPORT DiskFontBase,
                               DiskFontName;
FROM DOSLibrary         IMPORT DOSBase, DOSName;
FROM GraphicsLibrary    IMPORT GraphicsBase,
                               GraphicsName;
FROM Intuition          IMPORT IntuitionBase,
                               IntuitionName;
FROM LayersLibrary      IMPORT LayersBase,
                               LayersName;
FROM TranslatorLibrary IMPORT TranslatorBase,
                               TranslatorName;

TYPE

 LibraryShelf = ( Exec, DiskFont, Dos, Graphics,
                  Intuit, Layers, Translate);
 OnTheShelf   = SET OF LibraryShelf;

CONST

 FullShelf    = OnTheShelf { Exec, DiskFont, Dos,
                             Graphics, Intuit, Layers,
                             Translate };
 NormalShelf = OnTheShelf { Dos };

PROCEDURE RememberOpenLibraries () : OnTheShelf;
PROCEDURE OpenLibraries ( VAR LibraryCart :
                          OnTheShelf ) : BOOLEAN;
PROCEDURE CloseLibraries ( VAR LibraryCart :
                           OnTheShelf ) : BOOLEAN;
PROCEDURE AmigaOpenHouse ( VAR LibraryCart :
                           OnTheShelf ) : BOOLEAN;
PROCEDURE AmigaCloseHouse ( VAR LibraryCart :
                            OnTheShelf ) : BOOLEAN;

END AmigaHousekeeping.
```

Listato 3: AMIGA Housekeeping, implementazione.

```
IMPLEMENTATION MODULE AmigaHousekeeping;


FROM SYSTEM             IMPORT NULL;
FROM Libraries          IMPORT CloseLibrary,
                               OpenLibrary;
FROM AMIGAX             IMPORT ExecBase;
FROM DiskFontLibrary    IMPORT DiskFontBase,
                               DiskFontName;
FROM DOSLibrary         IMPORT DOSBase, DOSName;
FROM GraphicsLibrary    IMPORT GraphicsBase,
                               GraphicsName;
FROM Intuition          IMPORT IntuitionBase,
                               IntuitionName;
FROM LayersLibrary      IMPORT LayersBase,
                               LayersName;
FROM TranslatorLibrary IMPORT TranslatorBase,
                               TranslatorName;

(*
   I TYPE globali che si possono trovare nel modulo
   DEF sono ripetuti per motivi di chiarezza.
*)

(*
TYPE

 LibraryShelf = ( Exec, DiskFont, Dos, Graphics,
                  Intuit, Layers, Translate);
 OnTheShelf   = SET OF LibraryShelf;

CONST

 FullShelf    = OnTheShelf { Exec, DiskFont, Dos,
                             Graphics, Intuit, Layers,
                             Translate };
 NormalShelf = OnTheShelf { Dos };
*)

(* Fine dei TYPE globali del modulo DEF. *)

PROCEDURE RememberOpenLibraries () : OnTheShelf;

VAR
 LibraryBook : LibraryShelf;
 LibraryCart : OnTheShelf;

BEGIN (* RememberOpenLibraries *)

 (* Inizializzazione per la somma degli elementi. *)

 LibraryCart := OnTheShelf {};

(*
   Controlla ognuna delle librerie possibili per
   vedere se sono aperte.
*)

 FOR LibraryBook := Exec TO Translate DO
  CASE LibraryBook OF
   Exec      : IF ExecBase  NULL
               THEN (* aggiunge exec lib *)
                LibraryCart := LibraryCart +
                       OnTheShelf { Exec }
               END |

   DiskFont  : IF DiskFontBase  NULL
               THEN
```

```
                (* aggiunge disk font lib *)
                LibraryCart := LibraryCart +
                       OnTheShelf { DiskFont }
               END |

   Dos       : IF DOSBase  NULL
               THEN (* aggiunge dos lib *)
                LibraryCart := LibraryCart +
                       OnTheShelf { Dos }
               END |

   Graphics  : IF GraphicsBase  NULL
               THEN (* aggiunge graphics lib *)
                LibraryCart := LibraryCart +
                       OnTheShelf { Graphics }
               END |

   Intuit    : IF IntuitionBase  NULL
               THEN
                (* aggiunge intuition lib *)
                LibraryCart := LibraryCart +
                       OnTheShelf { Intuit }
               END |

   Layers    : IF LayersBase  NULL
               THEN (* aggiunge layers lib *)
                LibraryCart := LibraryCart +
                       OnTheShelf { Layers }
               END |

   Translate : IF TranslatorBase  NULL
               THEN
                (* aggiunge translator lib *)
                LibraryCart := LibraryCart +
                       OnTheShelf { Translate }
               END |

   ELSE (* Non fare niente *)
  END (* CASE *)
 END (* FOR *);

 RETURN LibraryCart

END RememberOpenLibraries;

PROCEDURE OpenLibraries ( VAR LibraryCart :
                       OnTheShelf ) : BOOLEAN;

VAR
 LibraryBook : LibraryShelf;

BEGIN (* OpenLibraries *)
 IF NOT ( LibraryCart = OnTheShelf {} )
  THEN
   FOR LibraryBook := Exec TO Translate DO
    CASE LibraryBook OF
     Exec      : IF Exec IN LibraryCart
                 THEN
                  ExecBase := OpenLibrary (
                      "exec.library", 0 );
                  LibraryCart := LibraryCart -
                         OnTheShelf { Exec }
                 END |

   DiskFont  : IF DiskFont IN LibraryCart
                THEN
                 DiskFontBase := OpenLibrary (
                     DiskFontName, 0 );
                 LibraryCart := LibraryCart -
                        OnTheShelf { DiskFont }
               END |

   Dos       : IF Dos IN LibraryCart
                THEN
                 DOSBase := OpenLibrary ( DOSName,
                    0 );
                 LibraryCart := LibraryCart -
                        OnTheShelf { Dos }
               END |

   Graphics  : IF Graphics IN LibraryCart
                THEN
                 GraphicsBase := OpenLibrary (
                         GraphicsName, 0 );
                 LibraryCart := LibraryCart -
                        OnTheShelf { Graphics }
               END |

   Intuit    : IF Intuit IN LibraryCart
                THEN
                 IntuitionBase := OpenLibrary (
                          IntuitionName, 0 );
                 LibraryCart := LibraryCart -
                        OnTheShelf { Intuit }
               END |

   Layers    : IF Layers IN LibraryCart
                THEN
                 LayersBase := OpenLibrary (
                       LayersName, 0 );
                 LibraryCart := LibraryCart -
                        OnTheShelf { Layers }
               END |

   Translate : IF Translate IN LibraryCart
                THEN
                 TranslatorBase := OpenLibrary (
                           TranslatorName, 0);
                 LibraryCart := LibraryCart -
                        OnTheShelf { Translate }
               END |

    ELSE (* non fare niente *)
   END (* CASE *)
  END (* FOR *);
  RETURN LibraryCart = OnTheShelf {}
 END (* IF *)

END OpenLibraries;

PROCEDURE CloseLibraries ( VAR LibraryCart :
                         OnTheShelf ) : BOOLEAN;
```

```
VAR
 LibraryBook : LibraryShelf;

BEGIN (* CloseLibraries *)
 IF NOT ( LibraryCart = OnTheShelf {} )
  THEN
   FOR LibraryBook := Exec TO Translate DO
    CASE LibraryBook OF

     Exec      : IF Exec IN LibraryCart
                 THEN
                  IF ExecBase  NULL
                   THEN CloseLibrary ( ExecBase )
                  END;
                  LibraryCart := LibraryCart -
                         OnTheShelf { Exec }
                END |

     DiskFont  : IF DiskFont IN LibraryCart
                  THEN
                   IF DiskFontBase  NULL
                    THEN CloseLibrary (DiskFontBase)
                   END;
                   LibraryCart := LibraryCart -
                          OnTheShelf { DiskFont }
                 END |

     Dos       : IF Dos IN LibraryCart
                  THEN
                   IF DOSBase  NULL
                    THEN CloseLibrary ( DOSBase )
                   END;
                   LibraryCart := LibraryCart -
                          OnTheShelf { Dos }
                 END |

     Graphics  : IF Graphics IN LibraryCart
                  THEN
                   IF GraphicsBase  NULL
                    THEN CloseLibrary (GraphicsBase)
                   END;
                   LibraryCart :=  LibraryCart -
                          OnTheShelf { Graphics }
                 END |

     Intuit    : IF Intuit IN LibraryCart
                  THEN
                   IF IntuitionBase  NULL
                    THEN CloseLibrary (
                              IntuitionBase )
                   END;
                   LibraryCart := LibraryCart -
                          OnTheShelf { Intuit }
                 END |

     Layers    : IF Layers IN LibraryCart
                  THEN
                   IF LayersBase  NULL
                    THEN CloseLibrary ( LayersBase )
                   END;
                   LibraryCart := LibraryCart -
                          OnTheShelf { Layers }
                 END |

     Translate : IF Translate IN LibraryCart
                  THEN
                   IF TranslatorBase  NULL
                    THEN CloseLibrary (
                              TranslatorBase )
                   END;
                   LibraryCart := LibraryCart -
                          OnTheShelf { Translate }
                 END |

      ELSE (* non fare niente *)
     END (* CASE *)
    END (* FOR *);
    RETURN LibraryCart = OnTheShelf {}
  END (* IF *)
END CloseLibraries;

PROCEDURE AmigaOpenHouse ( VAR LibraryCart :
                           OnTheShelf ) : BOOLEAN;

(* Normalmente la exec library e' gia' aperta. *)

VAR
 LibrariesOpened : OnTheShelf;

BEGIN (* AmigaOpenHouse *)
(* Ora apriamo le liberie di cui abbiamo bisogno. *)
 LibraryCart :=   FullShelf
                - LibraryCart
                - RememberOpenLibraries ()
                - OnTheShelf { Exec };

 (* Calcola le librerie che abbiamo aperto. *)

 LibrariesOpened := FullShelf - LibraryCart;

 RETURN OpenLibraries ( LibrariesOpened )

END AmigaOpenHouse;

PROCEDURE AmigaCloseHouse ( VAR LibraryCart :
                            OnTheShelf ) : BOOLEAN;

(*
   Normalmente la exec library viene chiusa
   automaticamente
*)

BEGIN (* AmigaCloseHouse *)
 RETURN CloseLibraries ( LibraryCart )
END AmigaCloseHouse;

BEGIN (* AmigaHousekeeping *)
END AmigaHousekeeping.
```

Quel Guru ha veramente un messaggio!

di Betty Clay

C'è qualcuno che sia mai riuscito a evitare il Guru? Nel momento cruciale vedete apparire il requester che dice:

```
"Software error - task held.
Cancel ALL disk activity.
Select CANCEL to reset/debug."
```

Quando premete il pulsante sinistro del mouse, appare il riquadro di allarme rosso e nero con il messaggio di "Guru Meditation":

```
"SOFTWARE FAILURE! PRESS LEFT MOUSE BUTTON TO
CONTINUE"
```

seguito da un numero composto da molte cifre, come ad esempio #03000007.000057A3 oppure #83010009.00002463. L'unica cosa che l'utente comune può fare è resettare il computer.

I messaggi sembrano totalmente privi di significato; a volte sembra persino uno scherzo crudele studiato appositamente dai creatori dell'Amiga per prendersi gioco degli utenti indifesi. Perchè?

In quelle meditazioni guru c'è veramente un messaggio. Tutti possiamo imparare il significato della maggior parte di esse se abbiamo le informazioni necessarie. I programmatori seri possono collegare un terminale all'Amiga con un cavo null modem, disporre i programmi workbench per l'operazione di debugging prima di caricare il programma, e lavorare sui programmi attraverso il terminale senza cambiare lo stato dell'Amiga. Possono utilizzare il programma di debugging ROMWACK che si trova su tutti i dischi KickStart, oppure il programma di debugging GRANDWACK, che fa parte dei pacchetti per programmatori, e possono consultare durante il loro lavoro i manuali ROM KERNEL. Questi manuali non sono scritti in quello che la maggior parte di noi chiamerebbe inglese, ma contengono più informazioni di quanto molti di noi non si sarebbero mai aspettati dalla Commodore, ed è lì che troviamo le spiegazioni dei messaggi Guru, chiamati più propriamente "Alert".

Lo scopo di questo articolo non è quello di spiegare come correggere gli errori di un programma utilizzando tutto il materiale necessario, bensì di dare un significato agli "alert" e far capire al lettore che vi sono dei programmatori per i quali questi messaggi sono veramente utili.

Prima di tutto dividiamo il messaggio in parti più facilmente gestibili. Poi cercheremo di dare un senso ad ogni parte. Prendete un messaggio Guru (ipotetico) come:

```
#07060006.00008321
```

Dividetelo come segue:

```
07   06   0006.   00008321
```

```
#07060006.00008321
```

La prima parte (07) indica da quale device o library viene il messaggio di errore. Esistono venti device, library, ecc. di questo tipo e sono elencati in ordine numerico nella pagina di riferimento di questo articolo. Osservando la tabella vedrete che "07" indica che l'errore è avvenuto durante l'uso della DOS library. Lo zero iniziale indica che è possibile porre un rimedio all'errore se si possiedono il materiale e la conoscenza appropriati per eseguire l'operazione di debugging. Per ora dobbiamo accontentarci di sapere che se la cifra iniziale è zero, l'errore può essere correggibile; una cifra iniziale pari a otto o più indica un errore al quale non si ha speranza di porre rimedio. Quindi potreste avere un errore nella DOS library che inizia con "07" (rimediabile) oppure con "87" (irrimediabile). Questi sono numeri esadecimali, quindi nella colonna di sinistra potreste trovare le lettere A, B, C, D o E.

Tecnicamente parlando, se l'errore è irreversibile, il bit più alto è a uno. Inoltre quando si hanno degli errori irreversibili tutto lo schermo diventa nero, mentre con quelli reversibili lo schermo viene solo spostato leggermente più in basso.

```
#0706 0006.00008321
```

La seconda informazione (06) può essere uno dei sei "codici di errore generico". In questo caso ho scelto l'errore "I/O" perché sembrava più adatto come errore del disco, sebbene in realtà è più probabile che l'errore si verifichi in altre library. Nella pagina di riferimento sono anche elencati i codici dei sei errori generici. Quello che incontrerete più frequentemente è "01", memoria insufficiente. Se l'errore generico non può essere attribuito a uno di questi sei codici, in questo campo del messaggio di errore dovrebbero esserci due zeri.

```
#0706 0006.00008321
```

Le quattro cifre che si trovano immediatamente a sinistra dei decimali indicano che cosa è andato storto effettivamente. Nel nostro esempio di messaggio le cifre sono "0006" e la tabella di riferimento mostrerà che, nella DOS library, il codice sei indica un errore di sequenza in un blocco del disco.

È probabile che non vi ricordiate questi numeri poiché lo stesso numero significherà cose diverse per le varie library o device. La tabella di riferimento include una lista di questi codici uguali a quelli che si trovano sulla versione 1.0 del disco WACK, un disco di debugging fornito ai programmatori. La

Commodore adesso ha cominciato a vendere questo disco, ma il prezzo sul mercato è di 99 dollari. Ricordate che le Guru Meditation sono state ideate per aiutare i programmatori, quindi le spiegazioni dei codici potrebbero essere inadeguate per coloro che hanno una conoscenza più scarsa, tuttavia le informazioni derivanti dal file "exec/alerts.h" danno molti indizi sull'errore che si sta verificando. Per saperne molto di più è necessario studiare il manuale ROM KERNEL.

```
#07060006.00008321
```

Le otto cifre che si trovano dopo i decimali danno un'informazione in numero esadecimale sull'indirizzo del task che ha segnalato l'errore. Poiché Amiga è una macchina che lavora in multitasking, è necessario che tutto il software sia rilocabile, e tali locazioni possono variare ogni volta che caricate il programma.

I codici elencati nella tabella sono stati suddivisi in settori per renderli più leggibili di quanto non siano nelle "Guru Meditation". Nelle prime due cifre troverete il numero di library o device; negli esempi qui riportati il codice di errore generico che si trova nella seconda coppia di cifre è sempre o "01" (memoria insufficiente) o "00", ed è stato fatto un tentativo di dare un seppur minimo significato al codice finale di quattro cifre. Notate, inoltre, che pochi di questi errori vengono definiti irreversibili. Molti errori nelle Library Exec, Graphics e Intuition sono irreversibili, molti altri non lo sono.

Qui di seguito riportiamo alcuni esempi:

```
#81000003.0000A325
81   - errore irreversibile nella Exec Library
00   - nessuno degli errori generici
0003 - errore nella checksum della library
.0000A325 - riferito al task che inizia nella
            locazione $A325

#21000001.00001234
21   - errore reversibile nel Disk Resource
00   - errore non generico
0001 - l'unità possiede già un disco
.00001234 - il task inizia nella locazione
            $1234

#84010005.00002345
84   - errore irreversibile nella Intuition
       Library
01   - errore causato da memoria insufficiente
0005 - necessità di aprire una nuova finestra
.00002345 - errore richiamato da un task
            nella locazione $2345
```

Questa informazione potrebbe far risparmiare ore a un abile programmatore che sia in possesso della letteratura e di una profonda conoscenza dell'Amiga.

Esiste un altro tipo di errore che può causare una "Guru Meditation". Tali errori sono caratterizzati da una sequenza di zeri, eccetto l'ultima cifra o le ultime due cifre che precedono i decimali. Si tratta di errori causati dallo stesso microprocessore 68000. Per esempio:

```
Meditation #00000002.xxxxxxxxx
```

...segnalerebbe che il microprocessore ha ricevuto un'istruzione non consentita. Per la maggior parte di noi è sufficiente sapere che se le prime sei o sette cifre sono degli zeri, l'errore è irreversibile ed è stato causato dal microprocessore 68000. Non possiamo porvi rimedio, ma possiamo riconoscere l'errore.

Per coloro che devono ancora imparare il C e l'assembler del 68000 è bene aspettare prima di effettuare delle correzioni, ma è rassicurante sapere che possiamo guardare il messaggio e dire a noi stessi: "Bene, ho rovinato i miei dati e ho perso il programma, ma almeno so che questo è causato dal fatto che la Graphics Library non aveva memoria sufficiente per disegnare il mio testo". Ciò può essere risolto aggiungendo ulteriore memoria. Con una conoscenza più approfondita anche gli altri problemi diventeranno risolvibili.

Annotazioni sulla tabella di riferimento

Vi sono pochissime Guru Meditation che non siano interpretabili con l'aiuto di queste tabelle. Una di queste è stata descritta da Claudio Nieder, uno studente di informatica in Svizzera. Continuava ad ottenere il numero di Guru Meditation #84000009.48454C50. Avendo notato che la parte dell'indirizzo aveva una forte somiglianza con il codice ASCII, lo ha tradotto:

```
48 45 4C 50
H  E  L  P
```

Ha ricevuto immediatamente una risposta da Dale Luck della Commodore-Amiga. Sembra che l'Amiga exec utilizzi la parola HELP come parola chiave. Quando Intuition si trova di fronte a un sistema così rovinato da non poter neppure visualizzare l'errore, inserisce la parola HELP nella locazione 0 e poi resetta il sistema. Nel momento in cui il sistema viene rilanciato, controlla se la parola HELP si trova nella locazione 0, e se così è, l'errore viene visualizzato.

Carolyn Scheppner, del Commodore Technical Support, recentemente ha studiato con cura alcuni dei messaggi Guru causati dallo stesso 68000. I due codici che si riferiscono al microprocessore e che appaiono più frequentemente sono 00000003 e 00000004. Il codice 3 (errore di indirizzo) può essere causato quando vengono passati a una routine di sistema puntatori non validi, oppure non più validi o equivalenti a zero, provocando in tal modo un tentativo di effettuare delle manipolazioni di word o longword in un indirizzo casuale. Il codice 4 (istruzione non consentita) può essere causato da una codifica insufficiente, ma può anche accadere quando la memoria contenente i codici o i vettori necessari è stata sovraccaricata.

Sembra che alcune delle exec.alert possano persino essere usate per diagnosticare problemi di hardware. Gli esempi dati dalla signora Scheppner erano 87000008 e 8700000B, che significano "key already free" e "key out of range". Lei dice che tali errori potrebbero riferirsi a problemi con la tastiera o con l'energia statica.

TABELLA DI RIFERIMENTO PER I MESSAGGI GURU

DEVICE, LIBRARY E RESOURCE:

01	Exec Library	08	RAM Library	15	Timer Device
02	Graphics Library	09	Icon Library	20	CIA Resource
03	Layers Library	10	Audio Device	21	Disk Resource
04	Intuition Lybrary	11	Console Device	22	Misc. Resource
05	Math Library	12	GamePort Device	30	BootStrap
06	List Library	13	Keyboard Device	31	Workbench
07	DOS Library	14	TrackDisk Device		

CODICI DI ERRORE GENERICO

01	Insufficient Memory	(Memoria insufficiente)
02	MakeLibrary Error	(Errore nella creazione di una library)
03	OpenLibrary Error	(Errore nell'apertura di una library)
04	OpenDevice Error	(Errore nell'apertura di un device)
05	OpenResource Error	(Errore nell'apertura di una Resource)
06	I/O Error	(Errore di Input/Output)

CODICI DI ERRORE SPECIFICO

EXEC LIBRARY: 01 00 0000

ExcptVect	81 00 0001	Il trap vector della CPU ha un checksum scorretto.
BaseChkSum	81 00 0002	ExecBase ha un errore di checksum
LibChkSum	81 00 0003	La library ha un errore di checksum
LibMem	81 00 0004	Memoria insufficiente per creare la library
MemCorrupt	81 00 0005	L'elenco della memoria libera è errato
IntrMem	81 00 0006	Memoria insufficiente per gli interrupt server.

GRAPHICS LIBRARY: 02 00 0000

CopDisplay	82 01 0001	Memoria insufficiente per la copper display list
CopInstr	82 01 0002	Memoria insufficiente per la copper instruction list
CopListOver	82 00 0003	Sovraccarico della Copper list (troppo lunga)
CopListHead	82 01 0005	Memoria insufficiente per la copper list head
LongFrame	82 01 0006	Memoria insufficiente per un LongFrame
ShortFrame	82 01 0007	Memoria insufficiente per un ShortFrame
FloodFill	82 01 0008	Memoria insufficiente per il FloodFill
TextTmpRas	02 01 0009	Memoria insufficiente per scrivere il testo
BltBitMap	82 01 000A	Memoria insufficiente per la BltBitMap

LAYERS LIBRARY: 03 00 0000

INTUITION LIBRARY: 04 00 0000

GadgetType	84 00 0001	Tipo di gadget sconosciuto, irreversibile
GadgetType	04 00 0001	Formula di recupero per un tipo di gadget
CreatePort	84 01 0002	Memoria insufficiente per creare una porta

ItemAlloc	84 01 0003	Allocazione di un oggetto sul piano, memoria insufficiente
SubAlloc	84 01 0004	Suballocazione, memoria insufficiente
PlaneAlloc	84 01 0005	Allocazione del piano, memoria insufficiente
ItemBoxTop	84 00 0006	Item Box topRelZero
OpenScreen	84 01 0007	Memoria insufficiente per aprire uno schermo
OpenScrnRast	84 01 0008	Allocazione del raster per l'apertura dello schermo, memoria insufficiente
SysScrnType	84 00 0009	Apertura di uno schermo sys, genere sconosciuto
AddSWGadget	84 01 000A	Add SW gadgets, memoria insufficiente
OpenWindow	84 01 000B	Memoria insufficiente per aprire la finestra
BadState	84 00 000C	Errato return con il comando interrupt
BadMessage	84 00 000D	Messaggio errato ricevuto dalle IDCMP (Intution Direct Communication Message Ports)
WeirdEcho	84 00 000E	Incomprensione causata da echo inusuale
NoConsole	84 00 000F	Impossibilità di aprire il Console Device

DOS LIBRARY: 07 00 0000

StartMem	07 01 0001	Troppo poca memoria durante lo startup
EndTask	07 00 0002	Il task non è terminato!
QPktFail	07 00 0003	Mancato funzionamento del QPkt (messaggio di coda?)
AsyncPkt	07 00 0004	Ricevuto messaggio inaspettato
FreeVec	07 00 0005	Mancato funzionamento di FreeVec
DiskBlkSeq	07 00 0006	Errore nella sequenza dei blocchi del disco
BitMap	07 00 0007	La bitmap non è valida
KeyFree	07 00 0008	La key è già libera
BadChkSum	07 00 0009	Il checksum non è valido
DiskError	07 00 000A	Errore del disco
KeyRange	07 00 000B	La key è fuori scala)
BadOverlay	07 00 000C	

TRACKSIDE DEVICE: 14 00 0000

TDCalibSeek	14 00 0001	Calibrate: ricerca l'errore
TDDelay	14 00 0002	Delay: errore nel wait del timer

DISK RESOURCE: 21 00 0000

DRHasDisk	21 00 0001	get unit: c'è già un disco
DRIntNoAct	21 00 0002	Interrupt: unità non attivata

WORKBENCH: 31 00 0000

CODICI DI ERRORE DELLA CPU:

02	Errore del Bus
03	Errore di indirizzo
04	Istruzione non consentita
05	Divisione per zero
06	Istruzione CHK (il risultato supera i limiti)
07	Trap overflow
08	Violazione nelle precedenze
09	Trace di istruzione
0A	Line A Emulation - Il 68000 ha trovato una parola di istruzione con valori tra A000 e AFFF
0B	Line F Emulation - Il 68000 ha trovato una parola di istruzione con valori tra F000 E FFFF

Le regole del gioco.

- Scopo del gioco è quello di infor[illegible]re, capire e divertirsi col computer;
- ogni giocatore deve evitare le cassette pirata, i game copiati e le riviste "vuote" di contenuti e di idee;
- ogni giocatore deve selezionare i propri acquisti. In edicola deve cercare sempre il marchio Jackson Software D.O.C., il marchio che è garanzia di sicurezza per cassette, riviste, floppy disk e computer game;
- ciascun computer vuole le cose migliori per giocare davvero bene e oggi solo Jackson Software D.O.C. garantisce l'alta qualità di ogni prodotto;
- quando il giocatore ha bisogno di informazioni, spiegazioni, consigli o idee può chiamare, con "filo diretto", le redazioni Jackson;
- il gioco, con Jackson Software D.O.C., non finisce mai e il divertimento è assicurato.
- Se ti è piaciuto, ricomincia a giocare.

GRUPPO EDITORIALE JACKSON

AREA CONSUMER

Scegli il meglio: scegli Jackson.

Come creare delle Run-Time library

Aggiungete funzioni personalizzate a quelle di sistema

di Steve Simpson

Steve Simpson è un consulente freelance di programmazione di Amiga e del PC IBM. In precedenza ha lavorato come astronomo, ma i suoi attuali interessi comprendono lo sviluppo di utility DOS e il controllo di processi industriali. Al momento si occupa dello sviluppo di routine per il SideCar. Nell'ambiente di Amiga è noto per aver scritto DiskRepair, una utility per l'editing e il salvataggio dei dischi malfunzionanti.

Chiunque abbia programmato su Amiga avrà senz'altro una certa familiarità con l'impiego delle librerie di run-time. Queste sono contenute sia nella ROM del Kickstart (tra queste le più note sono la graphics.library, la exec.library e la dos.library che contengono le routine relative alla grafica, al kernel dell'Exec e all'AmigaDOS) sia come file con il suffisso .library nella directory libs in un disco di sistema. Le run-time library costituiscono il cuore dell'architettura software di Amiga.

Una delle parti di un mio progetto è consistita proprio nella costruzione di una library in cui raggruppare le routine di uso più frequente, in modo che potessero essere accessibili dai miei programmi nel contesto del multitasking, evitando di tenere copie separate nei diversi moduli eseguibili. Questo articolo fornisce una descrizione dettagliata di come possa essere costruita una run-time library e di come possa essere aggiunta al sistema.

Cos'è una run-time library

Nell'architettura del sistema di Amiga, le routine di sistema e le jump-table non risiedono in locazioni fisse, ma vengono piuttosto caricate in memoria quando servono e in qualunque area della memoria in cui ci sia posto per esse. Questo si traduce in un sistema flessibile e in un uso efficiente della memoria, il che è particolarmente importante in un sistema multitasking. Inoltre, grazie all'assenza di indirizzi fissi, l'installazione di eventuali nuove release non comporta assolutamente alcun problema.

Notate che il termine "library" è usato nell'ambiente di programmazione di Amiga, con due differenti significati in due differenti contesti:

- un insieme di routine da usarsi al run-time contenute nel KickStart o caricate da disco e accessibili tramite delle jump-table in RAM.
- definizioni standard e routine varie impiegate durante il linking (alcune delle quali preposte proprio all'interfacciamento con le routine di run-time).

La figura 1 illustra la struttura basilare di una library; questa è costituita da una struttura Library e da una generica area per i dati, unite a una tavola di vettori di salto (generalmente ampia 6 bytes) alle routine della library. Queste routine non appaiono esplicitamente nella struttura principale della library, ma vengono caricate ovunque ci sia sufficiente spazio.

Figura 1: *La relazione tra le componenti di una run-time library e le routine della library stessa.*

La funzione OpenLibrary() attraversa una lista di nomi di library mantenuta dall'Exec e determina da questa l'indirizzo base della library. Tale indirzzo viene restituito nel registro D0; l'a-

rea dati della library si estende nelle locazioni successive.

Il nodo o struttura associata a una library è definito nel file "header" C col nome di exec/libraries.h nella maniera seguente:

```
struct Library
  {
  struct  Node lib_Node;
  UBYTE   lib_Flags;
  UBYTE   lib_pad;
  UWORD   lib_NegSize;
  UWORD   lib_PosSize;
  UWORD   lib_Version;
  UWORD   lib_Revision;
  APTR    lib_IdString;
  ULONG   lib_Sum;
  UWORD   lib_OpenCnt;
  };
```

I singoli elementi della struttura Library hanno il seguente significato:

lib_Node: una sottostruttura Node. Al suo interno, il campo ln_Name punta al nome della library, ln_Pri determina dove porre questa libreria nella lista delle library e ln_Type è posto a NT_LIBRARY per indicare che il nodo è relativo a una library.

lib_Flags: ovviamente, i flag della library; LIBF_SUMMING indica che un task sta al momento operando un checksum sulla library; LIBF_CHANGED indica che uno o più vettori della jump-table sono stati alterati; LIBF_SUMUSED indica che il progettista della library vuole che un fallimento del checksum porti ad un reset della macchina (cioè a un crash).

lib_NegSize: il numero di byte che precedono la struttura Library stessa. Corrisponde al numero di byte occupati dalla tavola dei vettori di salto.

lib_PosSize: il numero di byte che seguono la struttura Library e che costituiscono la area dati.

lib_Version: numero di versione della library.

lib_Revision: numero di revisione della library.

lib_IdString: puntatore a una stringa di testo terminata da un NULL che identifica univocamente la library.

lib_Sum: valore corrente del checksum della library.

lib_OpenCnt: è un contatore che viene incrementato ogni volta che la library viene aperta e decrementato ogni volta che viene chiusa.

A titolo di esempio, possiamo dare uno sguardo alla graphics.library. La sua jump-table incomincia nella maniera seguente:

```
offsets
FFC4 (-60) JMP $00FC7FF8    Text
FFCA (-54) JMP $00FC7F98    TextLength
FFD0 (-48) JMP $00FC7CC0    ClearScreen
FFD6 (-42) JMP $00FC7C88    ClearEOL
FFDC (-36) JMP $00FC7D24    BltTemplate
FFE2 (-30) JMP $00FC746A    BltBitMap
FFE8 (-24) JMP $00FC6CD4    ExtFunc
FFEE (-18) JMP $00FC6CD8    Expunge
FFF4 (-12) JMP $00FC6CD8    Close
FFFA (-6)  JMP $00FC6CD4    Open
0          Library Base Address
```

Gli indirizzi specificati nelle istruzioni di salto non sono necessariamente sempre gli stessi: essi dipendono, infatti, dall'area in cui l'AmigaDOS ha caricato la library. Comunque, il numero delle istruzioni di salto nella jump-table rimane ovviamente invariato. La jump-table si estende in senso inverso nella memoria (cioè da locazioni con indirizzo maggiore verso quelle a indirizzo minore) a partire dall'indirizzo base della library; si devono quindi impiegare offset negativi per individuare, tramite l'indirizzamento indiretto, l'istruzione di salto corrispondente alla funzione desiderata. Se l'indirizzo base della graphics.library, chiamato GfxBase, è posto nel registro A6, allora l'istruzione:

```
JSR -48(A6)
```

comporterà la pulizia dello schermo. In assembler, l'offset è definito come variabile globale nel sorgente oppure è definito come una costante nel file .i appropriato. A un offset definito in ambito globale viene aggiunto il prefisso _LVO (Library Vector Offset); per esempio, l'istruzione precedente può essere scritta come:

```
JSR _LVOClearScreen(A6)
```

Tali costanti di offset vengono definite in un file .i e seguono quelle delle routine standard di una generica library; si ha, cioè:

```
LIBINIT
  .....
  LIBDEF  CLEAREOL
  LIBDEF  CLEARSCREEN
  LIBDEF  TEXTLENGTH
  .....
```

dove LIBINIT definisce appunto gli offset delle routine standard (Open, Close, Expunge e ExtFunc). Quanto segue viene impiegato nel sorgente in assembler per saltare alla routine ClearScreen:

```
LINKLIB ClearScreen, A6
```

LINKLIB è una macro e nel caso in esame equivale a:

```
MOVE.L  A6,-(SP)
MOVE.L  A2,A6
JSR     -48(A6)
MOVE.L  (SP)+,A6
```

I listati che accompagnano questo articolo illustrano come accedere alle funzioni di una libreria e come conglobare routine in una run-time library.

Come costruire una run-time library

Il manuale ROM Kernel Manual: Libraries and Devices (RKM) fornisce un esempio di scheletro di una library, ma non descrive come implementarla né come renderla disponibile al sistema. Inoltre, in tali listati ci sono diversi gravi errori che li rendono del tutto inutili, come purtroppo ho scoperto a mie spese!

Il listato 1 fornisce il codice per mylib.library. Tale codice fa uso dei moduli "include" libsupp.i e mylib_def.i (rispettivamente i listati 2 e 3). La prima istruzione nella library pone a zero il registro D0 e ritorna immediatamente: ciò rappresenta una forma di protezione nel caso in cui qualcuno tenti erroneamente di eseguire la library.

La struttura ROM-Tag precisa al sistema dove sono contenute le routine e gli eventuali dati di inizializzazione. Il suo formato in C (definito nel file exec/resident.h) è:

```
struct Resident
  {
  UWORD   rt_MatchWord;
  struct  Resident *rt_MatchTag;
  APTR    rt_EndSkip;
  UBYTE   rt_Flags;
  UBYTE   rt_Version;
  UBYTE   rt_Type;
  BYTE    rt_Pri;
  char    *rt_Name;
  char    *rt_IdString;
  APTR    rt_Init;
  };
```

rt_MatchWord: posto sempre uguale a RTC_MATCHWORD (per identificare una struttura ROM-tag).

rt_MatchTag: puntatore alla prossima struttura ROM-tag.

rt_EndSkip: indirizzo del codice di terminazione della library.

rt_Flags: flag specifici di una ROM-tag: RTF_AUTOINIT o RTF_COLDSTART.

rt_Version: numero di versione.

rt_Type: tipo di ROM-tag: NT_LIBRARY, NT_DEVICE o NT_RESOURCE.

rt_Pri: la priorità della library; generalmente zero. Specifica la posizione desiderata nell'interno di lista delle library mantenuta dall'Exec.

rt_Name: puntatore alla stringa di nome terminata da un NULL. Nel nostro esempio "mylib.library\0".

rt_IdString: puntatore ad una stringa terminata da un NULL. Questa è un'etichetta specifica della library e della versione; risulta utile nello sviluppo e negli aggiornamenti, per identificare le eventuali diverse versioni di una library. Il suo formato è:

```
nome ver.rev (gg mmm aaaa),CR,LF,NULL
```

ad esempio:

```
"mylib 1.0 (06 May 1988)\015\012\0"
```

\015 e \012 sono gli equivalenti in ottale di CR (carriage return) e di LF (line feed)).

rt_Init: puntatore alla tavola di inizializzazione di questa libreria, cioè ad una tavola di indirizzi per l'inizializzazione dei dati e delle routine. Questa tavola viene definita in C con la seguente struttura:

```
struct Init
  {
  ULONG space;
  ULONG funcTable;
  ULONG dataTable;
  ULONG initRoutine;
  };
```

I campi della struttura Init hanno il seguente significato:

space: numero di byte dedicati all'area dati. Corrisponde alla somma di sizeof(struct Library) con la dimensione in byte dell'area dati utente.

funcTable: puntatore a una tavola di routine di inizializzazione o di indirizzi delle funzioni vere e prorie della library.

dataTable: puntatore a una tavola di inizializzazione dati.

initRoutine: puntatore alla routine che inizializza la library.

Questi parametri sono usati come argomenti per la funzione MakeLibrary() di cui si parlerà in seguito.

Nel listato 1, la parte di codice dopo funcTable, specifica gli indirizzi assoluti delle routine della library. Alle routine qui specificate verrà assegnato un posto nella jump-table. Generalmente, ad ogni routine corrispondono nella jump-table sei byte; si possono, però, avere istruzioni di salto anche di due o di quattro byte a seconda di quanto vicino stiano le routine alla struttura Library da cui dipendono. Se, ad esempio, le routine si trovano entro 32K dalla struttura Library, si possono impiegare nella jump-table istruzioni di salto lunghe quattro byte. Il termine della tavola è segnalato con -1L.

Se la prima long word della tavola è proprio -1L, allora i singoli elementi della tavola sono spiazzamenti relativi lunghi ognuno una word. Anche in questo caso -1L termina la tavola.

Sempre nel listato 1, la parte di codice seguente dataTable inizializza le strutture dati statiche. In questo esempio, noi inizializziamo solo la struttura Library con i valori che desideriamo, ma potete usare questa tavola per inizializzare le vostre aree

dati dopo averle allocate con AllocMem(). Il formato usato è lo stesso di quello impiegato con la funzione InitStruct(). Questa routine pulisce una zona di memoria e inizializza con i valori specificati le locazioni indicate nella tavola di inizializzazione. Le macro usate nel programma in assembler del listato 1 sono equivalenti alle seguenti dichiarazioni C:

```
struct InitByte
  {
  UBYTE ib_command;
  UBYTE ib_ccunt;
  UWORD ib_offset;
  BYTE  ib_value;
  UBYTE ib_pad;
  };

struct InitWord
  {
  UBYTE iw_command;
  UBYTE iw_count;
  UWORD iw_offset;
  UBYTE iw_value;
  };

struct InitLong
  {
  UBYTE il_command;
  UBYTE il_count;
  UWORD il_offset;
  LONG  il_value;
  };
```

Il formato del comando, che nelle tre strutture è il primo byte, è descritto nel RKM nelle pagine relative alla funzione InitStruct(). Per un esempio del suo uso ci si può riferire ancora al listato 1, dove vengono inizializzati diversi parametri nella struttura Library. Uno 0L indica la fine della tavola.

Dopo che MakeLibrary() ha allocato la library, viene chiamata la funzione InitRoutine() che consente inizializzazioni specifiche e personalizzate. Nel nostro esempio, vengono salvate copie del puntatore SysBase, del puntatore al segmento della library e del puntatore alla DOS library, dopo che essa è stata regolarmente aperta.

Interfaccia col sistema

Una library deve essere composta da un minimo di quattro funzioni standard: Open, Close, Expunge e ExtFunc. Quando l'utente chiama OpenLibrary(), CloseLibrary() o RemoveLibrary(), il sistema, in realtà, chiama proprio le routine Open, Close ed Expunge della library specificata. La funzione ExtFunc è stata dichiarata invece riservata.

Le routine create dall'utente sono poste dopo le summenzionate funzioni. Se si sviluppa una routine al di fuori della parte principale del codice della library, allora deve essere dichiarata "globale" (il che in assembler si ottiene tramite XREF <nome funzione>). Le routine possono essere scritte indifferentemente in C o in assembler (o in qualsiasi altro linguaggio) e vengono connesse al corpo principale nella fase di "linking".

Interfaccia di chiamata alle funzioni di una library

Convenzionalmente su Amiga si impiega il registro A6 come indirizzo base della library chiamata, mentre i registri D0, D1, A0 e A1 sono usati per passare i dati alle routine di una library; il risultato viene invece restituito generalmente in D0.

Questa modalità di scambio dei dati va bene se si programma in assembler; ma in C, i parametri delle funzioni chiamate vengono posti sullo stack. In tal caso si rende necessario estrarli dallo stack e porli ordinatamente nei registri, prima di saltare alla routine e ciò, purtroppo, tende a ridurre l'efficienza del codice C sotto questo punto di vista. Nel listato 6 viene illustrato un modo per realizzare l'interfaccia tra un programma in C e una routine appartenente a una library. Si noti che nel codice che funge da interfaccia si fà riferimento all'indirizzo base della library, che deve essere quindi già stato opportunamente inizializzato con OpenLibrary().

Le interfacce possono essere raggruppate in una libreria (il termine ha qui il senso specificato in precedenza con (2)) che va impiegata al momento del "linking". Nel nostro caso, le interfacce delle routine di esempio potrebbero essere conglobate in unico modulo oggetto my.lib, da impiegarsi come libreria con il linker.

Se, ora, le routine della libreria scritte in assembler si trovano a loro agio con gli argomenti nei registri, non si può dire lo stesso delle routine scritte in C. Il listato 7 si riferisce a tale situazione; all'ingresso della routine, il valore posto in A0 (che magari è stato estratto dallo stack dalla routine di interfaccia MyFunc2) viene posto sullo stack. Questo può essere realizzato con una sola istruzione assembler:

```
MOVE.L A0,-4(A5)
```

dove A5 contiene lo stack pointer attuale. In questo esempio, -4(A5) corrisponde alla variabile locale "val" nella routine __MyFunc2 (che nell'esempio C appare come _MyFunc2: il compilatore aggiunge un "underscore" a qualunque simbolo globale). Questa convenzione è specifica del compilatore ed è quindi indispensabile riferirsi alla documentazione relativa.

In questa maniera possiamo ora tranquillamente effettuare le operazioni che desideriamo e restituire il risultato in D0.

Come usare la library

La library creata dall'utente viene aperta nella maniera consueta, cioè con l'istruzione:

```
libBase = OpenLibrary(name,version);
```

dove: `struct Library *libBase;`

Il parametro "version" permette di richiedere l'apertura solamente di quelle library aventi numero di versione maggiore o al più uguale a quello desiderato. Nel listato 9 si mostra come aprire e chiudere correttamente una library e come chiamare le

routine della library che abbiamo creato.

Una volta che abbiamo terminato di usare la library, è opportuno che essa venga chiusa; ciò viene fatto con l'istruzione:

```
CloseLibrary(libBase);
```

Come installare una libreria nel sistema

Le parti che compongono una run-time library vanno assemblate o compilate con l'opzione che esclude la parte di codice iniziale di "startup" (che per chi usa il Manx si chiama .begin e per chi usa il Lattice si chiama startup.obj), che risulta inutile dal momento che ne forniamo uno esplicitamente noi stessi. Il file che otteniamo dopo il linking è la library vera e propria, che nel nostro esempio si chiama my.library.

Il listato 8 illustra come una library definita dall'utente venga aggiunta al sistema. La library viene dapprima caricata in memoria dalla funzione LoadSeg(); quindi, una chiamata a MakeLibrary() inizializza la jump-table, dispone appropriatamente le aree dati e opera l'inizializzazione come richiesto dall'utente. Tra i parametri di MakeLibrary() ci sono infatti un puntatore alla tavola degli indirizzi delle funzioni di libreria, una tavola per l'inizializzazione dei dati, un puntatore alla routine che inizializza la library, lo spazio in byte occupato dalla library e un puntatore BCPL alla "segment list" restituita da LoadSeg(). Il parametro che specifica lo spazio occupato si riferisce alla memoria occupata dalla struttura Library e dalla zona dati utente.

La chiamata a MakeLibrary() segue questa forma:

```
libBase = MakeLibrary(funcInit, structInit,
        libInit,dataSize,segList);

dove:  struct Library  *libBase;
```

Se il parametro structInit è NULL, non viene chiamata la funzione InitStruct(); ciò significa o che la library non si serve di una area dati o che comunque l'area dati viene inizializzata in qualche altra maniera. Se libInit è NULL, la routine Init non viene chiamata e la library non è quindi inizializzata nella maniera consueta.

Ci sono, poi, due altri modi per rendere una library disponibile al sistema. Questi metodi sono forse più semplici, ma affinché funzionino correttamente bisogna seguire strettamente il formato usato nel nostro esempio in assembler.

Quando si richiede l'apertura di una library con OpenLibrary(), il sistema esamina la lista delle library già installate e disponibili. Se non vi trova quella richiesta, la cerca allora nella directory specificata dal device logico dell'AmigaDOS LIBS:. Se a questo punto viene trovata, essa viene caricata, inizializzata, aperta e finalmente viene restituito il puntatore alla base della library.

Alternativamente, una library può essere salvata nella directory :Expansion del disco di sistema (il che è utile se avete un SideCar o una BridgeBoard), creando in aggiunta un file icona con il nome di <nome library>.info. Quando viene fatto il boot, le routine nella expansion.library si occupano proprio di caricare e inizializzare eventuali library con il suffisso .info.

Il debugging delle library

Questo paragrafo potrebbe, da solo, essere l'argomento di un intero articolo. Non si può, quindi, che fare una trattazione estremamente sommaria.

In primo luogo, bisogna fare attenzione alle convenzioni usate dal compilatore per passare i parametri dalle routine C a quelle assembler. Sotto questo aspetto, la possibilità di inserire codice assembler nel sorgente C può rivelarsi molto vantaggiosa.

Se vi è possibile, fate il debugging delle routine prima di inserirle nel contesto della library. In ogni caso, un buon debugger, specialmente se a livello di sorgente, torna sempre utile.

Assicuratevi, poi, che l'ordine che attribuite alle routine nella tavola delle funzioni sia lo stesso di quello specificato nel file assembler in cui viene definito LIBINIT (nell'esempio mylib_def.i).

Conclusioni

Il concetto di run-time library migliora l'efficienza del sistema, consentendo ai programmi di usare un insieme standard di funzioni, evitando inutili copie. Questo articolo dovrebbe aver fatto luce su un aspetto del sistema di Amiga per il quale non esiste al momento una soddisfacente documentazione.

L'esempio che compare nei listati è stato sviluppato con l'assembler e compilatore C Manx Aztec C68K v3.60a e corretto con l'ausilio del debugger a livello di sorgente Manx SDB.

Bibliografia

ROM Kernel Reference Manual: Libraries and Devices, Addison-Wesley

ROM Kernel Reference Manual: Exec, Addison-Wesley

The Kickstart Guide to the Amiga: Ariadne Software Ltd.

Commodore Amiga Inc., *The AmigaDOS Manual*, 2nd edition, Bantam

Mortimore, E., *Amiga Programmers Guide*, Vol. 1, Sybex

Williams, S., *Programming the 68000*, Sybex

Listato 1:

```
;-----------------------------------------------
;
; mylib.asm - codice della libreria di esempio
;
; Esempio di come si costruisce una run-time library
;
; Storia:
;    88-05-06  creato da Steve Simpson
;                        Skarpskyttev0C
;                        S-222 42 Lund
;                        Sweden
;
; assemblaggio: as mylib.asm
;
; link: ln -o mylib.library mylib.o _myfunc2.o c.lib
;
;-----------------------------------------------

 SECTION     section

 NOLIST
 INCLUDE "exec/types.i"
 INCLUDE "exec/nodes.i"
 INCLUDE "exec/lists.i"
 INCLUDE "exec/libraries.i"
 INCLUDE "exec/alerts.i"
 INCLUDE "exec/resident.i"
 INCLUDE "exec/initializers.i"
 INCLUDE "libraries/dos.i"
 INCLUDE "libsupp.i"
 INCLUDE "mylib_def.i"
 LIST

;-----------------------------------------------
; definizioni e riferimenti esternu
;-----------------------------------------------
 XDEF     Init            ;inizializzazione library
 XDEF     Null
 XDEF     myName
 XDEF     Open            ;routine standard
 XDEF     Close           ;  --"--
 XDEF     Expunge         ;  --"--
 XDEF     ExtFunc         ;  --"--
 XDEF     MyFunc0         ;routine dell'utente
 XDEF     MyFunc1         ;  --"--
 XDEF     __MyFunc2       ;  --"--  (questa e' in C)

 XREF     _AbsExecBase    ;l'unico indirizzo fisso
                           in un sistema Amiga

 XLIB     OpenLibrary
 XLIB     CloseLibrary
 XLIB     Alert
 XLIB     FreeMem
 XLIB     Remove

;-----------------------------------------------
; La prima locazione eseguibile. Restituisce un
; errore nel caso che qualcuno cerchi di eseguire
; la libreria come se fosse un programma.
;-----------------------------------------------
LibStart:
         CLEAR    d0
         rts

;-----------------------------------------------
; una serie di definizioni ed EQU
;-----------------------------------------------
MYPRI    EQU      0   ;priorita' della libreria
VERSION  EQU      1   ;numero maggiore di versione
REVISION EQU      1   ;revisione particolare. Deve
                      ;identificare univocamente la
                      ;libreria

;-----------------------------------------------
;Una struttura romtag.
;Sia exec che ramlib cercano questa struttura
;durante le operazioni di caricamento per scoprire,
;per esempio, da che punto partire.
;In C, questo è equivalente alla struttura Resident.
;-----------------------------------------------
initLibDescrip:                ;stuttura RT,0
         dc.w     RTC_MATCHWORD  ;UWORD RT_MATCHWORD
         dc.l     initLibDescrip ;APTR  RT_MATCHTAG
         dc.l     EndLibCode     ;APTR  RT_ENDSKIP
         dc.b     RTF_AUTOINIT   ;UBYTE RT_FLAGS
         dc.b     VERSION        ;UBYTE RT_VERSION
         dc.b     NT_LIBRARY     ;UBYTE RT_TYPE
         dc.b     MYPRI          ;UBYTE RT_PRI
         dc.l     myName         ;APTR  RT_NAME
         dc.l     libIdString    ;APTR  RT_IDSTRING
         dc.l     libInit        ;APRT  RT_INIT
                                 ;LABEL RT_SIZE

myName          MYLIBNAME          ;my library name

;-----------------------------------------------
;identificatore della libreria usato per supportare
;la libreria.
;il format e':
           'nome versione revisione (dd mon yyyy)',
           <cr>,<lf>,<null>
;-----------------------------------------------

libIdString    dc.b     'mylib 1.0 (06 May 1988)',
13,10,0

dosName        DOSNAME    ;nome della dos library

         ds.w     0        ;allineamento su word

;-----------------------------------------------
;romtag specifica "RTF_AUTOINIT". Il campo RT_INIT
;punta alla tabella seguente. Se RTF_AUTOINIT non e'
;impostato allora RT_INIT punta a una routine di
;inizializzazione che deve essere esguita.
;Questi dati sono usati dal programma di caricamento
```

```
;come parametri ad AddLibrary().
;-------------------------------------------------
libInit:
        dc.l    MyLib_Sizeof        ;dimensione
dei dati
        dc.l    funcTable           ;putantatore agli
                                    ;inizializzatori
                                    ;delle funzioni
        dc.l    dataTable           ;putantatore agli
                                    ;inizializzatori
                                    ;dei dati
        dc.l    initRoutine         ;routine da
                                    ;eseguire

;-------------------------------------------------
;tabella degli indirizzi delle funzioni di mylib
;-------------------------------------------------
funcTable:
    ;routine standard - DEVONO sempre esserci
        dc.l    Open
        dc.l    Close
        dc.l    Expunge
        dc.l    ExtFunc
    ;definitzioni per la mia libreria
        dc.l    MyFunc0
        dc.l    MyFunc1
        dc.l    __MyFunc2
    ;marcatore di fine tabella
        dc.l    -1

;-------------------------------------------------
;questa tabella inizializza le strutture dati. Il
;formato e' specificato nelle pagine del manuale
;relative a exec/InitStruct. Il primo argomento e'
;l'offset rispetto alla base della libreria. Il
;secondo argomento e' il valore da porre in quella
;cella. La tabella e' terminata con un null.
;-------------------------------------------------
dataTable:
        INITBYTE    LH_TYPE,NT_LIBRARY
        INITLONG    LN_NAME,myName
        INITBYTE    LIB_FLAGS,
                    LIBF_SUMUSED!LIBF_CHANGED
        INITWORD    LIB_VERSION,VERSION
        INITWORD    LIB_REVISION,REVISION
        INITLONG    LIB_IDSTRING,idString
        dc.l        0

;-------------------------------------------------
;Questa e' la routine di inizializzazione - viene
;chiamata dopo che la libreria e' stata allocata.
;Se restiruisce un valore diverso da zero, la
;libreria verra' collocata nella lista delle
;librerie alla posizione indicata dalla priorita'.
;-------------------------------------------------
initRoutine:        ; D0: lib. ptr., A0:segment list
    ;mette il puntatore alla libreria in A5
        move.l      a5,-(sp)
        move.l      d0,a5

    ;salva il puntatore a exec
        move.l      a6,ml_SysLib(a5)

    ;salva un puntatore alla segment list
        move.l      a0,ml_SegList(a5)

    ;apre dos.library
        lea         dosName(pc),a1
        CLEAR       d0
        CALLSYS     OpenLibrary

        move.l      d0,ml_DosLib(a5)
        bne.s       1$

    ;non puo' aprire dos.library
        ALERT       AG_OpenLib!AO_DOSLib

1$:
    ;genera i dati statici di cui ha bisogno
    ;
    ;le inizializzazioni specifiche delle librerie
    ;devono essere messe qui.
    ;

        move.l      a5,d0
        move.l      (sp)+,a5
        rts

;-------------------------------------------------
;I comandi di interfacciamento del sistema iniziano
;qui. Le chiamate a OpenLibrary/CloseLibrary/
;RemoveLibrary vengono tradotte in chiamate alle
;routine Open/Close/Expunge. Il puntatore alla base
;della libreria e' in A6. Il task switching e' stato
;disabilitato (via Forbid/Permit).
;-------------------------------------------------

;-------------------------------------------------
;Open restituisce il puntatore alla libreria in D0
;se l'apertura ha avuto successo, altrimenti ritorna
;un null.
;-------------------------------------------------
Open:     ;libptr: A6,  version: D0
    ;incrementa il contatore delle aperture
    ;della libreria
        addq.w      #1,LIB_OPENCNT(a6)

    ;evita expunge ritardati
        bclr        #LIBB_DELEXP,ml_Flags(a6)

        move.l      a6,d0
        rts

;-------------------------------------------------
;Se la libreria non e' piu' aperta, Close ritorna la
;segment list se il flag di expunge ritardato e'
;impostato; altrimenti resituisce NULL.
;-------------------------------------------------
Close:    ;(libptr: a6)
    ;setta il valore di ritorno
        CLEAR       d0
```

```
    ;decrementa il contatore delle aperture
    ;della libreria
        subq.w      #1,LIB_OPENCNT(a6)

        bne.s       1$

    ;c'e' qualche expunge ritardato?
        btst        #LIBB_DELEXP,ml_Flags(a6)
        beq.s       1$

    ;esegue l'expunge
        bsr         Expunge

1$:
        rts

;-------------------------------------------------
;Se la libreria non e' piu' aperta, Expunge ritorna
;la segment list del codice della libreria.
;Altrimenti viene impostato il flag di expunge
;ritardato e viene restituito NULL in D0.
;-------------------------------------------------
Expunge:    ;libptr: A6
        movem.l     d2/a5/a6,-(sp)
        move.l      a6,a5
        move.l      ml_SysLib(a5),a6

        tst.w       LIB_OPENCNT(a5)
        beq         1$

    ;Se e' ancora aperta, imposta il flag
        bset        #LIBB_DELEXP,ml_Flags(a5)
        CLEAR       d0
        bra.s       Expunge_End

1$:
    ;Qui possiamo rimuovere noi stessi
        move.l      ml_SegList(a5),d2

    ;unlink dalla lista delle librerie
        move.l      a5,a1
        CALLSYS     Remove

    ;================================
    ;Chiusure specifiche della library
    ;================================

    ;chiude la dos library
        move.l      ml_DosLib(a5),a1
        CALLSYS     CloseLibrary

    ;libera la memoria
        CLEAR       d0
        move.l      a5,a1
        move.w      LIB_NEGSIZE(a5),d0

        sub.l       d0,a1
        add.w       LIB_POSSIZE(a5),d0

        CALLSYS     FreeMem

    ;imposta il valore di ritorno - la seglist
        move.l      d2,d0

Expunge_End:
        movem.l     (sp)+,d2/a5/a6
        rts

;-------------------------------------------------
;ExtFunc restituisce semplicemente 0 in D0.
;-------------------------------------------------
 ExtFunc:
        CLEAR       d0
        rts

;-------------------------------------------------
;I comandi specifici della libreria iniziano qui.
;In questa libreria di esempio ci sono 3 routine
;specifiche. La terza, __MyFunc2, e' scritta in C.
;-------------------------------------------------

;-------------------------------------------------
; MyFunc0 - restituisce semplicemente 0 in D0.
;-------------------------------------------------
 MyFunc0:
        CLEAR       d0
        rts

;-------------------------------------------------
; MyFunc1 - restituisce il contenuto del registro A6
;           nel registro D0
;-------------------------------------------------
 MyFunc1:
        move.l      a6,d0
        rts

;-------------------------------------------------
;Questa e' la fine del codice della libreria.
;-------------------------------------------------
 EndLibCode:
        END
```

Listato 2:

```
;-------------------------------------------------
; libsupp.i
;
; macro aggiuntive
;
; Storia:
;    88-05-06  creato
;-------------------------------------------------

CLEAR           MACRO
    moveq    #0,\1
    ENDM
```

```
LINKSYS         MACRO
    LINKLIB  _LVO\1,\2
    ENDM

CALLSYS         MACRO
    CALLLIB  _LVO\1
    ENDM

XLIB            MACRO
    XREF     _LVO\1
    ENDM
```

Listato 3:

```
;-------------------------------------------------
; mylib_def.i
;
; define assembly per la libreria mylib.library
;
; Storia:
;    88-05-06  creato
;-------------------------------------------------

;--offset delle funzioni dalla base della libreria
LIBINIT
 LIBDEF     MYFUNC0
 LIBDEF     MYFUNC1
 LIBDEF     MYFUNC2

;--area della struttura dati mylib
 STRUCTURE   MyLib,LIB_SIZE
   ULONG  ml_SysLib
   ULONG  ml_DosLib
   ULONG  ml_SegList
   UBYTE  ml_Flags
   UBYTE  ml_pad
   LABEL  MyLib_Sizeof

;--nome della mia libreria
MYLIBNAME          MACRO
      dc.b     'mylib.library',0
      ENDM
```

Listato 4:

```
;-------------------------------------------------
;  myfunc0.asm
;
; Interfaccia tra il programma chiamante e la jump
; table di mylib.library
; routine MyFunc0
;
; Storia:
;    88-05-06  creato
;-------------------------------------------------

 NOLIST
 INCLUDE "exec/types.i"
 INCLUDE "exec/libraries.i"
 INCLUDE "libsupp.i"
 INCLUDE "mylib_def.i"
 LIST

 XREF  _MyBase
 XREF  _MyFunc0

;-------------------------------------------------
; sequenza di chiamata C:  ret = MyFunc0();
;              registri:   D0
;-------------------------------------------------

_MyFunc0:
    ;--mette la base della libreria in A2
        move.l   _MyBase,a2

    ;--effettua la chiamata della routine
 LINKLIB MYFUNC0,a2

        rts

        END
```

Listato 5:

```
;-------------------------------------------------
;  myfunc1.asm
;
; Interfaccia tra il programma chiamante e la jump
; table di mylib.library
; routine MyFunc1
;
; Storia:
;    88-05-06  creato
;-------------------------------------------------

 NOLIST
 INCLUDE "exec/types.i"
 INCLUDE "exec/libraries.i"
 INCLUDE "libsupp.i"
 INCLUDE "mylib_def.i"
 LIST

 XREF  _MyBase
 XREF  _MyFunc1

;-------------------------------------------------
; sequenza di chiamata C:  ret = MyFunc1();
```

```
;               registri:   D0
;-----------------------------------------------------
_MyFunc1:
    ;--mette la base della libreria in A2
        move.l    _MyBase,a2

    ;--salta alla jump table della library
 LINKLIB MYFUNC1,a2

      rts

      END
```

Listato 6:

```
;
;-----------------------------------------------------
;  myfunc2.asm
;
; Interfaccia tra il programma chiamante e la jump
; table di mylib.library
; routine MyFunc2
;
; Storia:
;    88-05-06  creato
;-----------------------------------------------------

 NOLIST
 INCLUDE "exec/types.i"
 INCLUDE "exec/libraries.i"
 INCLUDE "libsupp.i"
 INCLUDE "mylib_def.i"
 LIST

 XREF  _MyBase
 XREF  _MyFunc2


;-----------------------------------------------------
; sequenza di chiamata C:  ret = MyFunc2();
;               registri:   D0
;-----------------------------------------------------

_MyFunc2:

    ;--mette nel registro il valore di input preso
    ;  dallo stack
        move.l   4(sp),a0

    ;--mette in A2 la base
        move.l   _MyBase,a2

    ;--ora salta alla jump table della library
 LINKLIB MYFUNC2,a2

        rts

        END
```

Listato 7:

```
/************************************************
 * _myfunc2.c
 *
 * Routine C (__MyFunc2) installata nella library.
 * Poiche' una chiamata a una routine di libreria in
 * assembler mette i dati dentro i registri,
 * dobbiamo riprendere i dati e metterli in
 * variabili locali per i programmi C.
 *
 * Storia:
 *  88-05-06  creato
 *
* compilare con:  cc +b _myfunc2
 *        Non abbiamo bisogno dello startup in
 *        questo modulo
 ************************************************/

#include <exec/types.h>

/************************************************
 *
 * Chiamata assembler:   registro di input: a0
 *
 ************************************************/

ULONG _MyFunc2()
{
  ULONG val;

  /* direttive assembler */
#asm
  ;--questa e' l'interfaccia alla libreria

      move.l   a0,-4(a5)
#endasm

    /* esegue un calcolo di esempio */
  val = val * 2;

    /* restituisce il valore che finisce in d0 */
  return(val);

} /* fine di _MyFunc2 */
```

Listato 8:

```
/*************************************************
 * add_lib.c
 *
 * Modulo che aggiunge una libreria definita dallo
 * utente alla lista delle librerie mantenuta dal
 * sistema. Dopo l'aggiunta la libreria e' a
 * disposizione di tutti i programmi
 *
 * Storia:
 *   88-05-06  creato
 *
 * compilare con:  cc add_lib
 * link con:       ln add_lib -lc
 *************************************************/

#include <functions.h>
#include <exec/types.h>
#include <exec/libraries.h>
#include <exec/resident.h>
#include <libraries/dos.h>
#include <libraries/dosextens.h>
#include <mylib.h>

ULONG segList,    /* puntatore BCPL alla seglist */
      codeloc;    /* puntatore al codice */

struct Init {
  ULONG space;    /* dimensione dello spazio dati */
  ULONG funcTable;    /* puntatore alla tabella
                         degli offset delle funzioni */
  ULONG dataTable;    /* ptr.alla table initStruct*/
  ULONG initRoutine;  /* ptr. a initRoutine */
} *init;

struct Resident *libRes;  /* puntatore all'area
                             romtag della libreria */

ULONG space, funcTable, dataTable, initRoutine;

struct MyLibrary *mylib;


main()
{

   /* carica il segmento della library */
  segList = (ULONG)LoadSeg("mylib.library");
  if (segList==NULL) {
    printf("Non posso caricare mylib.library\n");
    exit(0);
  }

   /* indirizzo dell'inizio del codice della
    * libreria moltiplicato per quattro perche'
    * si tratta di un puntatore BCPL */
  codeloc = segList * 4;

   /* indirizzo dell'area residente rom tag */
  libRes = (struct Resident *)(codeloc + 8);

   /* indirizzo della tabella di inizializzazione */
  init = (struct Init *)libRes->rt_Init;

   /* estrae i dati dalla tabella */
  space = init->space;
  funcTable = init->funcTable;
  dataTable = init->dataTable;
  initRoutine = init->initRoutine;

   /* inizializza la libreria */
  mylib = (struct MyLibrary *)MakeLibrary(funcTable,
          dataTable, initRoutine, space, segList);

   /* rende nota la libreria al sistema */
  AddLibrary(mylib);

  printf("mylib.library aggiunta al sistema\n");

}
```

Listato 9:

```
/*************************************************
 * my_main.c
*
 * Programma di prova per mylib.library
 *
 * Storia:
 *   88-05-06  creato da Steve Simpson
 *
 * compilare con:  cc my_main
 * link con:       ln my_main -lmylib -lc
 *************************************************/

#include <functions.h>
#include <exec/types.h>
#include <exec/libraries.h>
#include <mylib.h>

struct MyLibrary *MyBase;

main()
{
  ULONG ret, val;

    /* Apertura della libreria */
  MyBase = (struct MyLibrary *)
           OpenLibrary("mylib.library",1L);
  if (MyBase==NULL) {
    printf("Non posso aprire mylib.library\n");
    exit(0);
  }

    /* func0 resetta D0 e ritorna il suo valore */
```

```
  ret = MyFunc0();
  printf("ret(0)=%lx\n",ret);

    /* func1 mette A6 in D0 e lo restituisce */
  ret = MyFunc1();
  printf("ret(1)=%lx\n",ret);

    /* func2 moltiplica val per 2 e restituisce
     * il risultato */
  val = 10L;
  ret = MyFunc2(val);
  printf("ret(2)=%ld\n",ret);

    /* chiusura della libreria */
  CloseLibrary(MyBase);

}
```

Listato 10:

```
/*************************************************
 * mylib.h
 *
 * define per l'uso con mylib.library
 *
 * Storia:
 *    88-05-06  creato
 *************************************************/

#include <exec/types.h>
#include <exec/libraries.h>
#include <exec/execbase.h>
#include <libraries/dosextens.h>

/* struttura di mylib library- queata e' la versione
 * C di quanto definito in mylib_def.i
 */
struct MyLibrary {
  struct Library     ml_MyLib;
  struct ExecBase    *ml_SysLib;
  struct DosLibrary  *ml_DosLib;
  APTR               ml_SegList;
  UBYTE              ml_Flags;
  UBYTE              ml_pad;
};

#define MYLIBNAME  "mylib.library"
```

segue da pag. 22

Come richiamare il CLI...

```
IF p<>0 THEN
  WHILE p<>0
   File$=x$(p)
   PRINT
   Com$="DIR >ram:temp "+File$+" opt d"+z$
   x=Execute&(SADD(Com$), 0, Hand1& )
   x$(0)=File$
   OPEN "ram:temp" FOR INPUT AS #1
   n=0
   WHILE NOT EOF(1)
     LINE INPUT #1,x$
     n=n+1
     x$(n)=File$+MID$(x$,6,LEN(x$)-11)+"/"
   WEND
   CLOSE #1
   KILL "ram:temp"
   IF n>0 THEN
    PRINT 0;File$;"   ... oppure,"
    FOR j=1 TO n
     PRINT j;x$(j)
    NEXT j
    p=-1
    WHILE p<0 OR p>n
     PRINT "La vostra scelta (da 0 a";n;")";
     INPUT p
    WEND
   ELSE
    p=0
   END IF
   File$=x$(p)
  WEND
  PRINT
  PRINT "Stiamo per creare una nuova directory"
  PRINT "nell'area   ";File$
  INPUT "OK (s/n) ";x$
  IF x$="s" OR x$="si" THEN
   INPUT "Nome della nuova directory ";Nam$
   OPEN "ram:temp" FOR OUTPUT AS #1
     PRINT #1,"MakeDir ";File$;Nam$
   CLOSE #1
   PRINT "Creazione directory ";File$;Nam$
   ' Ora usiamo uno script file...
   Hand2&=xOpen&(SADD("ram:temp"+z$),1005)
   IF Hand2&<>0 THEN
    x=Execute&(SADD(z$), Hand2&, Hand1& )
    xClose(Hand2&)
   ELSE
    PRINT "Fallito!  Non posso gestire l'i/o"
   END IF
  END IF
  KILL "ram:temp"
  FILES File$
 END IF
xClose(Hand1&)
LIBRARY CLOSE
```

PER IL TUO COMPUTER

A. Bigiarini - P. Cecioni - M. Ottolini

IL MANUALE DI AMIGA

Rivolto soprattutto ai programmatori, per saperne di più e conoscere meglio i tre modelli di Amiga e le loro ampie possibilità. Poichè vengono presentate le differenze fra i tre modelli disponibili della macchina, il libro risulta utile anche come una funzionale guida all'acquisto.

SOMMARIO

Caratteristiche generali - Grafica - Sprite - Coprocessori - Audio - Interfacciamento - Chip 8520 - Compatibilità IBM - Rom Kemel - Amiga DOS 1.1 e 1.2 - Registri dei Chip Custom - SuperDOS - ARC - SNOOP 1.0.

244 pagine Cod. CZ532 L. 39.000

R. Bonelli - M. Lunelli

AMIGA 500

GUIDA PER UTENTE

Finalmente un testo in grado di racchiudere in un'unica guida tutte le informazioni necessarie agli utenti di Amiga 500, in modo che possano comprendere tutte le possibilità del loro sistema e utilizzarlo al meglio.

SOMMARIO

Uso del mouse - Uso dei menu - Programmi del disco Workbench - Programmi del disco extras - Amiga Dos - Amiga Basic - Il Basic compilato: AC BASIC - Il True Basic.

370 pagine Cod. CC627 L. 55.000

M. England - D. Lawrence

AMIGA HANDBOOK

Un libro per conoscere l'Amiga, il nuovo computer della COMMODORE, al fine di comprendere e sfruttare al massimo tutte le potenzialità di questo sistema considerato da molti rivoluzionario.

SOMMARIO

Uno sguardo all'Amiga - Chip 68000 - Copper co-processor - Playfield e sprite - Blitter - Comunicazioni con il mondo esterno - Nucleo e Exec - Sistema operativo - Workbench e le tecniche di intuition - DOS e Command line interface - Programmi in BASIC.

204 pagine Cod. CC320 L. 35.000

RITAGLIATE E SPEDITE IN BUSTA CHIUSA

GRUPPO EDITORIALE JACKSON
Via Rosellini, 12 - 20124 MILANO

INDICARE CHIARAMENTE CODICI E QUANTITÀ DEI VOLUMI RICHIESTI							
Codice	Q.tà	Codice	Q.tà	Codice	Q.tà	Codice	Q.tà

L. 4.000 per contributo fisso spese di spedizione

MODALITÀ DI PAGAMENTO

☐ Allego assegno n. _____ di L. _____ della Banca _____

☐ Ho effettuato il pagamento di L. _____ a mezzo: ☐ vaglia postale ☐ vaglia telegrafico ☐ versamento sul c/c postale n. 11666203 intestato a Gruppo Editoriale Jackson SpA Milano e allego fotocopia della ricevuta.

☐ Pagherò al postino l'importo di L. _____ al ricevimento dell'opera.

☐ Richiedo l'emissione della fattura (formula riservata alle aziende) e comunico il numero di Partita IVA _____

DATA _____ FIRMA _____

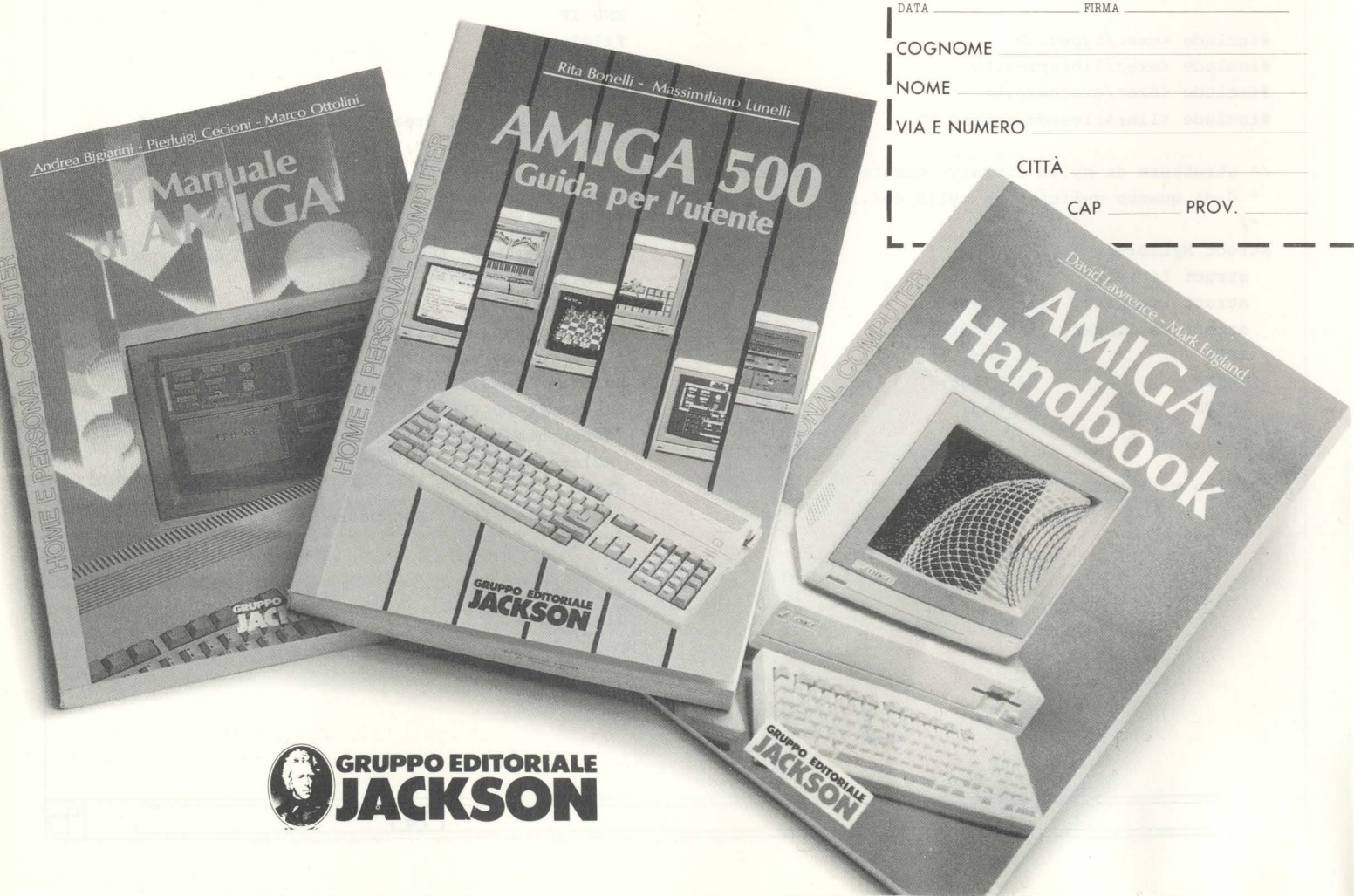
COGNOME _____

NOME _____

VIA E NUMERO _____

CITTÀ _____

CAP _____ PROV. _____



GRUPPO EDITORIALE JACKSON

Programmare in M2 Amiga

Tradurre il sorgente C

di Anthony Bryant

Anthony Bryant è un laureato della Università di Manitoba. È un ingegnere informatico che lavora con sistemi di controllo remoto e robotica; trascorre il suo tempo libero programmando Amiga.

Sia i programmatori professionisti che gli hacker della domenica hanno una cosa in comune: approfondire le proprie conoscenze. Lo studiare un nuovo linguaggio, per esempio, può offrire molti nuovi spunti che potranno essere utili nell'arte della programmazione. Questo articolo tratta alcune delle mie esperienze nell'adattare/tradurre sorgente C (tutto nel Pubblico Dominio) al sistema di sviluppo M2AMIGA.

Dal momento che il Modula-2 (d'ora in poi identificato come M2) è un linguaggio più recente del C, la maggior parte delle librerie di supporto (che sono la linfa vitale di Amiga) saranno adattamenti del sorgente C originale. Fortunatamente, M2AMIGA aderisce alla nomenclatura C nelle librerie, semplificando la traduzione. Il fatto che si possano fare compilazioni separate di moduli durante lo sviluppo e che si abbia a disposizione un compilatore rapido e un linker, permette veloci traduzioni.

Uno dei primi moduli di libreria su cui ho lavorato era il ben noto "AudioTools" di Rob Peck, una completa collezione di routine audio semplici da utilizzare che permettevano all'utente di accodare suoni (PlayFreq, PlayNote e PlaySamp) e suonarli, sequenzialmente, mentre il programma continuava a fare ciò che stava facendo. Rob mi inviò la sua versione più recente scritta in C, Release 3.0, e io mi misi all'opera. Arrivai rapidamente ad apprezzare il sorgente ben documentato (che sopravvive nella versione M2 - vedi AudioTools.def e AudioTools.imp). Non solo è importante conoscere che cosa fa il codice, ma anche che cosa aveva in mente l'autore nel momento in cui l'ha scritto. Il sorgente di Rob è anche un eccellente insegnamento sull'utilizzo di IORequest con i device.

Modifiche richieste.

Dal momento che il compilatore M2AMIGA produce codice in un passo, tutti gli oggetti referenziati debbono già essere noti. Se la dichiarazione di una procedura che deve essere chiamata giace più in basso nel programma, la procedura deve essere dichiarata come FORWARD (FORWARD è una parola riservata). Io ho optato, invece, per riordinare tutte le procedure cosicché questo non fosse necessario, e durante il processo di stesura di tutte le procedure chiamate da altre procedure, ottenni una visione della struttura del programma che si rivelò utile nell'aggiungere/modificare alcune routine.

Il passo successivo era di controllare la interfaccia con tutte le routine della libreria di Amiga; in questo caso, da "Dos" ed "Exec". M2 non permette procedure che siano sia funzioni che procedure. Per esempio, in C potreste avere:

```
error = WaitIO(iob); /* una funzione
                        procedura */
```

oppure

```
WaitIO(iob);          /* una procedura vera */
error = iob-iob-Request.io_Error;
```

Entrambi gli esempi restituiscono l'errore che è stato piazzato nel campo io_Error dopo che è terminata WaitIO. Ma in M2 l'ultimo caso è l'unica possibilità. Inoltre, si potrebbe incontrare in C:

```
dummy = WaitPort(uport);
```

mentre in M2, WaitPort è una procedura vera e propria, così che non vi è alcun bisogno di una variabile *dummy*, cioè una variabile di cui non ci interessa il contenuto.

Flag, operatori e costrutti.

La rappresentazione dei flag in M2 è sottile, paragonata al C. I tipi fondamentali, SHORTSET, BITSET, e LONGSET gestiscono l'insieme dei bit di flag in 8 bit, 16 bit e 32 bit rispettivamente. Ad esempio, i flag IORequests sono chiamati con il nome IOFlagSet (definiti come un SHORTSET; cioè, i bit da 0 a 7) con:

```
quick = IOFlagSet{0};cioè il bit 0 è a 1
noWait= IOFlagSet{6};cioè il bit 6 è a 1
```

Gli operatori + e - vengono utilizzati per assegnare o ripulire i flag. In realtà M2 non dispone degli operatori su bit del C e

non sussiste nemmeno la necessità. Potete controllare con facilità se un bit è assegnato oppure vuoto. Inoltre, potete definire i vostri SET come sottoinsiemi dei tipi fondamentali.

M2 mette a disposizione una buona selezione di istruzioni per cicli fra le quali scegliere. Decisioni, decisioni! In C, il costrutto per il ciclo 'for' è in realtà una variazione del costrutto base 'while'. Nell'adattare dal C, ho scoperto che, in molti casi, se la variabile indice del 'for' si incrementa o decrementa dopo, avrei scelto l'istruzione FOR del M2, ma se la variabile indice del 'for' si incrementa o decrementa prima, avrei scelto l'istruzione WHILE del M2. Il 'do...while' del C diventa il REPEAT...UNTIL del M2, ma non dimenticate di rovesciare la logica della espressione 'while'! Il solo uso di un 'goto' in AudioTools è stato gestito con facilità dalla combinazione LOOP ed EXIT del M2.

Operazioni con le stringhe.

La definizione delle stringhe in M2 è leggermente diversa da quella del C. Se volete effettuare il passaggio di un parametro stringa a una routine della libreria di Amiga, quella stringa deve avere un carattere nullo (in M2 sia 0C che "") come terminatore. Ma una variabile definita in M2, ad esempio:

```
VAR:
  string: ARRAY [0..11] OF CHAR;
```

potrebbe essere riempita completamente con 12 caratteri e nessun carattere nullo. Se sono meno di 12 allora, naturalmente, ci sarà un carattere nullo di terminazione. Una caratteristica speciale del compilatore M2AMIGA assicura che tutte le stringhe costanti, immagazzinate nello spazio delle costanti all'interno di un programma, vengano chiuse in modo appropriato da un carattere di terminazione. Pertanto:

```
OpenDevice(ADR("audio.device"), ..... )
```

oppure

```
if CONST audioName="audio.device" then
  OpenDevice(ADR(audioName), ..... )
```

funzionano altrettanto bene.

M2AMIGA fornisce le procedure richieste per la manipolazione delle stringhe (Compare(), Insert(), Delete(), Occurs(), ecc.) gestendo con facilità gli assegnamenti di stringhe.

Operazioni numeriche e indirizzi.

Anche le costanti numeriche vengono maneggiate con semplicità. Tutte le costanti, essendo prive di un tipo, vengono convertite (forzate) nel tipo richiesto dalla routine di libreria.

Gli interi aritmetici costituiscono un'altra area dove esistono sottili differenze. Le routine di Rob per calcolare la durata delle note e dei cicli necessari si prendono già cura di massimizzare la precisione facendo attenzione a non superare una parola di 32 bit. In M2, uno short privo di segno viene chiamato CARDINAL mentre un long privo di segno viene chiamato LONGCARD. A causa del rigoroso controllo sui tipi del M2, è fantastico moltiplicare o dividere gli stessi tipi. Non vi permetterà di moltiplicare un intero con segno, INTEGER oppure LONGINT, per un tipo privo di segno. Se sapete (oppure sospettate) che il prodotto di due CARDINAL sia un LONGCARD (molto probabile!) è meglio che operiate una conversione di tipi. M2AMIGA esegue conversioni di tipi piuttosto che trasferire i tipi [vocabolario M2] dal momento che è più sicuro. Se non lo fate, il run-time di M2AMIGA vi avviserà con un errore di "Overflow"!

Una volta che avete terminato la fase di controllo, potete eliminare il controllo fornito dal run-time. Questo fa in modo che il compilatore termini di generare frammenti di codice che eseguano un controllo sui sottoinsiemi, sugli indici degli array, sull'overflow e underflow, sulla divisione per zero, sull'overflow dello stack e altri ancora.

M2AMIGA definisce gli indirizzi come puntatori a un byte.

```
TYPE
  ADDRESS = POINTER TO BYTE;
```

L'indirizzo è un tipo compatibile con LONGINT, così quando de-referenziate un indirizzo state esaminando un byte, e quando incrementate o decrementate un indirizzo, è per mezzo del byte.

Ma in C, potete avere (BYTE *) e (WORD *), entrambi indirizzi, ma che differenza! Senza alcuna eccezione, potete tradurre (BYTE *) come indirizzo, e gli array di byte vengono utilizzati all'interno di AudioTools per conservare i dati delle forme d'onda dell'audio.

L'utilizzo di (WORD *) è più complesso, dal momento che è solitamente accompagnato dalla indicizzazione [] di un indirizzo. L'indicizzazione è per WORD. Il linguaggio C supera tutto ciò, sperando che il programmatore C sappia quello che sta facendo (indicizzare un indirizzo, addirittura!). In M2, si possono passare array di word aperti, che è proprio quello che stiamo trattando in questo caso, tra procedure. L'uso degli array aperti semplifica questo compito. L'array non viene duplicato, dal momento che questo ovviamente consumerebbe più spazio; invece, solo il suo indirizzo e la sua lunghezza vengono passati, assicurando l'integrità dei dati.

Il rigoroso controllo sui tipi del M2 vi costringe a fare mente locale su quanto state facendo e sui risultati delle vostre azioni; piuttosto che lasciar immaginare al compilatore ciò che voi intendevate. Porta via tutte (o almeno la maggior parte) delle ragnatele ed è meno probabile che veniate assaliti in seguito da un bug nascosto.

Gran parte del lavoro di adattamento, nonostante ciò, è stato di semplice editazione: ad esempio, sostituire '/*' e '*/' con '*' e '*', cambiare '=' in ':=' e '==' in '=', sostituire '!=' in '#' (personalmente preferisco il carattere # piuttosto che <>) e cambiare '-' in '^'. M2 dispone di comandi speciali, INC e DEC, per rimpiazzare '++', '+=', '--' e '-='. SHIFT gestisce '<<' e '>>', MOD corrisponde a '%', NOT è '!', AND è '&&', OR è '||' (mi torna alla mente l'articolo *Easy C* pubblicato sul

numero di Maggio 1986 di Byte). Fatto! Beh, quasi...

La caratteristica di cui ho maggiormente sentito la mancanza è la direttiva #define del compilatore C che, oltre a definire costanti, viene molto spesso utilizzata per rimpiazzare intere procedure, sia funzioni che procedure vere e proprie. All'interno di AudioTools, questa tecnica viene effettivamente impiegata per aggiungere tutti i comandi Channel, che sono stati definiti in termini di modifiche del comando base ControlChannel(). Non rimane altra scelta che scrivere un certo numero di procedure molto brevi.

Il modulo di AudioTools (su disco) ha la apparenza e la funzione di ogni altro modulo di libreria; è costituito da un file di simboli (sym/AudioTools.sym) che il compilatore utilizza, e un file oggetto (obj/AudioTools.obj) che il linker utilizza. Una delle ragioni per le quali M2AMIGA compila e effettua link così rapidamente è la presenza di questi moduli "pre-compilati". Tutti i file di simboli necessari vengono portati nella RAM durante la compilazione, e tutti i file oggetto necessari vengono allo stesso modo portati in RAM durante la fase di link.

Per verificare il funzionamento del programma, AudioDemo.mod, è un semplice esempio che importa tutto dalle procedure di AudioTools. Allo scopo di avere qualche forma d'onda da eseguire, ho settato dapprima alcuni array d'onda - dente di sega, triangolare e rettangolare. Sebbene M2AMIGA abbia ampie librerie matematiche in REAL, LONGREAL, trascendenti e IEEE trascendenti in doppia precisione, ho scelto il tipo speciale fast floating point del M2AMIGA, FFP, e ho simulato i miei seni e coseni. La conversione da FFP a INTEGER è abbastanza semplice ed ecco allora un array sinusoidale. Ho perfino verificato il tutto utilizzando un programmino in AmigaBasic, che otteneva un grafico utile per vedere che l'inizio della forma d'onda fosse allo stesso livello della fine, in qualche modo importante nell'effettuare il link dei dati delle onde audio per evitare rumori di ogni genere. AudioTools espande l'array di base di 256 byte allo scopo di raggiungere frequenze più elevate e queste onde vengono linkate end-to-end; vedi SetWave() all'interno di AudioTools.imp.

L'utilizzo del modulo di libreria Terminal di M2AMIGA semplifica notevolmente un breve programma di prova come questo. Dal momento che M2AMIGA supporta sia il CLI che il Workbench, Terminal apre automaticamente una finestra (se chiamato dal Workbench) nella quale scrivere stringhe (oppure scrivere numeri decimali o esadecimali). Il run-time di sistema di M2AMIGA nasconde un po' del lavoro del programmatore eseguendo lo startup, aprendo e chiudendo automaticamente le librerie di Amiga, e passando argomenti CLI oppure Workbench. Ciò potrebbe essere è un limite se necessitate assolutamente di un AStartup non-standard.

Il modulo libreria Arts di M2AMIGA forma la base di tutti i programmi ed è integrato con gli altri moduli di libreria (non i moduli di libreria Amiga). Quando avvengono degli errori di run-time durante la fase di sviluppo, questi non danno origine a Guru meditation. Al contrario, il programma viene interrotto e viene visualizzato un requester che riporta la causa dell'errore e tende a liberare tutte le risorse allocate che conosce attraverso gli altri moduli di libreria (tramite l'utilizzo di TermProcedure(), vedi oltre). Ad esempio, la terminazione dopo la chiusura del requester che visualizza l'errore, chiude la finestra Terminal (se era aperta), il modulo FileSystem chiude tutti i file aperti, il modulo Heap dealloca tutta la memoria che era stata precedentemente allocata, ecc.

Il modulo Arts contiene svariate procedure interessanti, come ad esempio BreakPoint(), la quale provoca una (temporanea) interruzione del programma, e DebugProcedure() che chiama un debugger, non implementato, in seguito a una interruzione o a un crash del programma. Potete addirittura aggiungere informazioni che debbono essere visualizzate all'interno del requester che riporta gli errori oppure creare il vostro debugger utilizzando le informazioni riguardanti interruzioni/crash contenute nella variabile errorFrame del modulo Arts.

Ovviamente, non dovete utilizzare nulla di tutto ciò (nessuno vi costringe) e pertanto, dal momento che AudioTools utilizza principalmente i moduli di libreria Amiga, ho deciso di utilizzare solo la TermProcedure() del modulo Arts per eseguire la mia pulizia, che ho deciso di chiamare FinishAudio(). TermProcedure() funziona al termine del programma e in caso di errore. Una volta terminata la fase di debugging, può effettivamente essere rimossa, purché non vi dimentichiate di chiamare FinishAudio()! Questo evita la necessità di effettuare un reset di Amiga, dopo ogni fase di debug.

Per finire, il codice generato per produrre l'eseguibile, AudioDemo, è altrettanto compatto quanto il suo equivalente scritto utilizzando il Manx C, sebbene contenga ulteriori aggiunte (la finestra Terminal, e così via).

AudioTools costituisce una valida aggiunta alle librerie in vostro possesso, sia in C sia in M2, e grazie al sorgente di Rob ho imparato molte cose su come lavorare direttamente sulle periferiche di Amiga. Dal momento che nella mia libreria possedevo PlaySamp e PlayNote, ho rivolto la mia attenzione a ReadSamp e ReadSmus, reperibili nelle specifiche IFF 8SVX e SMUS. Ma questa è tutta un'altra faccenda (nonchè una libreria molto più estesa).

Listato 1:

```
DEFINITION MODULE AudioTools;
(* ------------------------------------------------
 * Basato su AudioTools versione 3.0 di Rob Peck
 *
 * Questo software è nel pubblico dominio
 *
 * M2AMIGA Modula-2 adattamento di Anthony Bryant
 *------------------------------------------------*)

FROM SYSTEM IMPORT
 ADDRESS, BYTE;
FROM Audio IMPORT
 hardChannels;
FROM Exec IMPORT
 Byte, IORequest, Message, MsgPortPtr;

CONST
 maxChan = hardChannels;
```

```
 audBuffers = 20;
 audClock = 3579545;

 badChannelSelected = -1;
 notYourChannel     = -2;
 outOfMemory        = -3;

TYPE
 ExtIOB=RECORD
  request: IORequest;
  allocKey: INTEGER;
  data: ADDRESS;
  length: LONGCARD;
  period: CARDINAL;
  volume: CARDINAL;
  cycles: CARDINAL;
  writeMsg: Message;
  identifier: LONGINT;
 END;
 ExtIOBPtr=POINTER TO ExtIOB;

PROCEDURE InitAudio(): MsgPortPtr;
PROCEDURE CheckIfDone(): BOOLEAN;
PROCEDURE FinishAudio(uport: MsgPortPtr);

PROCEDURE GetChannel(channel: LONGINT): LONGINT;
PROCEDURE FreeChannel(channel: LONGINT): LONGINT;
PROCEDURE StopChannel(channel: LONGINT): LONGINT;
PROCEDURE StartChannel(channel: LONGINT): LONGINT;
PROCEDURE FlushChannel(channel: LONGINT): LONGINT;
PROCEDURE ResetChannel(channel: LONGINT): LONGINT;
PROCEDURE IsThatMyChan(channel: LONGINT): LONGINT;

PROCEDURE MayGetNote(uport: MsgPortPtr;
                     flag: BOOLEAN): LONGINT;

PROCEDURE SetWave(channel: LONGINT;
            VAR waveform: ARRAY OF BYTE): LONGINT;

PROCEDURE SetSamp(channel: LONGINT;
             VAR sampleaudio: ADDRESS;
                 length: LONGINT): LONGINT;

PROCEDURE SetPV(channel: LONGINT;
              period, volume: CARDINAL): LONGINT;

PROCEDURE PlayNote(channel: LONGINT;
                   note:    CARDINAL;
                   volume:  CARDINAL;
                   duration: CARDINAL;
                   priority: Byte;
                   messageport: MsgPortPtr;
                   id: LONGINT);

PROCEDURE PlayFreq(channel: LONGINT;
                   freq:    CARDINAL;
                   volume:  CARDINAL;
                   duration: CARDINAL;
                   priority: Byte;
                   messageport: MsgPortPtr;
                   id: LONGINT);

PROCEDURE PlaySamp(channel: LONGINT;
                   period:  CARDINAL;
                   volume:  CARDINAL;
                   duration: CARDINAL;
                   priority: Byte;
                   messageport: MsgPortPtr;
                   id: LONGINT);

END AudioTools.def
```

Listato 2:

```
IMPLEMENTATION MODULE AudioTools;
(* ------------------------------------------
 * Basato su AudioTools versione 3.0 di Rob Peck
 *
 * Questo software è nel pubblico dominio
 *
 * M2AMIGA Modula-2 adattamento di Anthony Bryant
 *------------------------------------------*)

FROM SYSTEM IMPORT
 ADDRESS,ADR,BYTE,LONGSET;
FROM Audio IMPORT
 free, perVol, allocate,
 pervol, syncCycle, noWait, writeMessage,
 IOAudio;
FROM Dos IMPORT
 Delay;
FROM Exec IMPORT
 invalid,reset,read,write,update,clear,stop,start,
 flush,quick,
 IORequest, Message, MsgPortPtr, Node, TaskPtr,
 UnitPtr, DevicePtr,
 UByte, Byte, MemReqs, MemReqSet,
 AllocMem, CloseDevice, FindTask, FreeMem, GetMsg,
 OpenDevice, PutMsg, WaitIO, WaitPort;
FROM ExecSupport IMPORT
 BeginIO, CreatePort, DeletePort;

CONST
 waveSize=512;

TYPE
 auMsg=RECORD
  message: Message;
  identifier: LONGINT;
 END;
 auMsgPtr=POINTER TO auMsg;

VAR
 unit: ARRAY [0..maxChan-1] OF UnitPtr;
 key:  ARRAY [0..maxChan-1] OF INTEGER;
 usertask: ARRAY [0..maxChan-1] OF TaskPtr;

 openIOB: IOAudio;
 device: DevicePtr;
 controlPort: MsgPortPtr;

 audbuffer: ARRAY [0..audBuffers-1] OF ExtIOB;
 inuse:     ARRAY [0..audBuffers-1] OF BOOLEAN;
```

```
  chipaudio: ARRAY [0..maxChan-1] OF ADDRESS;
  datalength: ARRAY [0..maxChan-1] OF LONGINT;
  replyPort: ARRAY [0..maxChan-1] OF MsgPortPtr;
  dynamix:   ARRAY [0..maxChan-1] OF LONGINT;
  anychan:   ARRAY [0..maxChan-1] OF UByte;

  dynamicName: ADDRESS;
  globalName: ADDRESS;

  woffsets:ARRAY [0..8] OF CARDINAL;
  wlen:    ARRAY [0..8] OF CARDINAL;
  perval:  ARRAY [0..12] OF CARDINAL;

(*------ procedure interne ----------------------*)

PROCEDURE FreeIOB(iob: ExtIOBPtr; channel: LONGINT);
 VAR
   i: CARDINAL;
 BEGIN
   IF (iob^.request.message.node.name = dynamicName)
     THEN FreeMem(iob, SIZE(iob^));
     IF (dynamix[channel] # 0) THEN
       DEC(dynamix[channel]);
     END;
 ELSIF (iob^.request.message.node.name = globalName)
     THEN i:= iob^.request.message.length;
     IF (i < audBuffers) THEN
       inuse[i]:= FALSE;
     END;
   END;
END FreeIOB;

(*-----------------------------------------------*)

(* ReEmployIOB - esamina TUTTE le ReplyPort e se c'è
in giro qualche IOB senza niente da fare, allora le
libera.

Audio potrebbe ancora essere impegnato a suonare la
forma d'onda quando riceviamo un messaggio
attraverso MayGetNote. MayGetNote marca i messaggi
iob come potenzialmente liberi (quando trova che il
campo identifier è messo a zero) ma noi dobbiamo
avere un modo per ricircolare in questa lista di
messaggi.

In altre parole, se qualcosa è potenzialmente
libero, liberalo, altrimenti lascialo nella lista.
Così, invece di rimuovere i messaggi dalla cima
della lista, esaminiamo tutta la catena rimuovendo
quello che è potenzialmente libero e lasciando il
resto per la prossima volta.
*)

PROCEDURE ReEmployIOB();
 VAR
   i: LONGINT;
   mp: MsgPortPtr;
   iob: ExtIOBPtr;
   pushback: ExtIOBPtr;
   (* Quello che succede qui è che le iob vengono
   rimosse dalla MessagePort quando ritornano
   indietro dall'audio device. Se NOI abbiamo
   settato la MessagePort con un valore diverso da
   zero, vuole dire che vogliamo essere informati
   quando la nota comincia a essere suonata. La
   parte WriteMsg di iob viene spedita, sotto forma
   di messaggio, alla nostra user port. Quindi
   questa routine non può liberare l'iob fino a
   quando non è sicura che NOI abbiamo finito di
   usarla. Qui viene letto il campo iob_Priority. Se
   è ancora diverso da zero, l'iob viene rispedita
   indietro alla message port (in fondo alla catena
   di messaggi) per essere esaminata nuovamente. Noi
   teniamo un puntatore chiamato "pushback" che ci
   permette di riconoscerla quando la incontriamo di
   nuovo. Quando la esaminiamo per la seconda volta,
   vuole dire che abbiamo fatto un giro completo
   attraverso la catena di messaggi e che per questa
   volta abbiamo liberato tutto quello che potevamo.
   Quindi la esaminiamo e la liberiamo o la
   rimandiamo ancora una volta indietro e poi
   usciamo.
   *)
 BEGIN
   FOR i:=0 TO maxChan-1 BY 1 DO
     mp:= replyPort[i];

     pushback:= NIL;
     iob:= ExtIOBPtr(GetMsg(mp));
     WHILE (iob # NIL) DO

       IF (iob^.writeMsg.replyPort # NIL) THEN
         PutMsg(mp, iob);
         IF ((iob # pushback) AND (pushback = NIL))
           THEN pushback:= iob;
         END;
       ELSE
         FreeIOB(iob, i);
       END;

       iob:= ExtIOBPtr(GetMsg(mp));

     END;  (* while *)
   END;  (* for *)
END ReEmployIOB;

(*-----------------------------------------------*)

(* GetIOB - alloca una struttura IOB, globale o
 * dinamica.
 *)

PROCEDURE GetIOB(channel: LONGINT): ExtIOBPtr;
 VAR
   i, usereply: CARDINAL;
   iob: ExtIOBPtr;
 BEGIN
   ReEmployIOB();
```

```
    IF (channel = -1) THEN usereply:= 0;
    ELSE usereply:= channel; END;

    FOR i:=0 TO audBuffers-1 BY 1 DO
      IF (inuse[i] = FALSE) THEN
        inuse[i]:= TRUE;
        audbuffer[i].request.device:= device;
        audbuffer[i].request.message.replyPort:=
                                  replyPort[usereply];
        audbuffer[i].request.message.length:= i;
        audbuffer[i].request.message.node.name:=
                                          globalName;
        RETURN ADR(audbuffer[i]);
      END;
    END;

    iob:= ExtIOBPtr(AllocMem(SIZE(iob^),
                              MemReqSet{memClear}));
    IF (iob = NIL) THEN RETURN NIL; END;

    iob^.request.device:= device;
    iob^.request.message.replyPort:=
                                  replyPort[usereply];
    iob^.request.message.node.name:= dynamicName;
    iob^.request.message.length:= dynamix[usereply];
    INC(dynamix[usereply]);
    RETURN iob
END GetIOB;

(*-----------------------------------------------*)
PROCEDURE CheckIOBDone(): BOOLEAN;
 VAR
   i, status: LONGINT;
 BEGIN
   status:= 0;

   FOR i:=0 TO audBuffers-1 BY 1 DO
     IF (inuse[i] = TRUE) THEN
       ReEmployIOB();
     END;
   END;

   FOR i:=0 TO maxChan-1 BY 1 DO
     IF (dynamix[i] > 0) THEN
       ReEmployIOB();
     END;
   END;

   FOR i:=0 TO maxChan-1 BY 1 DO
     IF (dynamix[i] = 0) THEN INC(status); END;
   END;

   IF (status = 4) THEN
     FOR i:=0 TO audBuffers-1 BY 1 DO
       IF (inuse[i] = TRUE) THEN RETURN FALSE; END;
     END;
     RETURN TRUE;
   ELSE
     RETURN FALSE;
   END;
END CheckIOBDone;

(*---------Procedure di supporto----------------*)

(* InitAudio - restituisce uport, un puntatore alla
message port alla quale il nostro task riceve un
messaggio quando una particolare nota inizia a
essere suonata. Dobbiamo salvare questo valore da
qualche parte per poterlo utilizzare chiamando
MayGetNote o FinishAudio. MayGetNote è il nome della
routine che chiamiamo per controllare se una nota è
in fase di esecuzione o meno. Se si verifica un
errore (OpenDevice o CreatePorts non funzionano per
qualche motivo) il puntatore restituito = NIL.
*)

PROCEDURE InitAudio(): MsgPortPtr;
 VAR
   error, i: LONGINT
   firstuser: BOOLEAN;
 BEGIN
   firstuser:= TRUE;

   FOR i:=0 TO audBuffers-1 BY 1 DO
     inuse[i]:= FALSE;
   END;

   openIOB.length:= 0;
   OpenDevice(ADR("audio.device"), 0, ADR(openIOB),
                                        LONGSET(0));

   error:= LONGINT(openIOB.request.error);
   IF (error # 0) THEN RETURN NIL; END;
   device:= openIOB.request.device;

   FOR i:=0 TO maxChan-1 BY 1 DO
     replyPort[i]:= CreatePort(0, 0);
     IF (replyPort[i] = NIL) THEN RETURN NIL; END;

     chipaudio[i]:= 0;
     datalength[i]:= 0;
     dynamix[i]:= 0;

     IF (firstuser = TRUE) THEN
       key[i]:= 0;
       unit[i]:= NIL;
       usertask[i]:= NIL;
     END;
   END;
   controlPort:= CreatePort(0, 0);
   IF (controlPort = NIL) THEN RETURN NIL; END;

   anychan[0]:= 1;
   anychan[1]:= 2;
   anychan[2]:= 4;
   anychan[3]:= 8;

   woffsets[0]:= 0;
   woffsets[1]:= 256;
   woffsets[2]:= 384;
   woffsets[3]:= 448;
   woffsets[4]:= 480;
```

```
    woffsets[5]:= 496;
    woffsets[6]:= 504;
    woffsets[7]:= 508;
    woffsets[8]:= 510;

    perval[0]:= 428;
    perval[1]:= 404;
    perval[2]:= 381;
    perval[3]:= 360;
    perval[4]:= 339;
    perval[5]:= 320;
    perval[6]:= 302;
    perval[7]:= 285;
    perval[8]:= 269;
    perval[9]:= 254;
    perval[10]:= 240;
    perval[11]:= 226;
    perval[12]:= 214;

    dynamicName:= ADR("dynamic");
    globalName:= ADR("global");

    RETURN CreatePort(0, 0);
END InitAudio;

(*-----------------------------------------------*)
(* GetChannel - per richiedere qualsiasi canale,
assegnare channel = -1; Per richiedere un canale
specifico, assegnare channel = 0, 1, 2 o 3; Notate
che questa funzione restituisce due globali oltre al
numero del canale!  *)

PROCEDURE GetChannel(channel: LONGINT): LONGINT;
 VAR
    error, channum: LONGINT;
    addrmsg: ADDRESS;
    iob: ExtIOBPtr;
    controlIOB: ExtIOB;
 BEGIN
    iob:= ADR(controlIOB);
    iob^.request.device:= device;
    iob^.request.message.replyPort:= controlPort;
    iob^.allocKey:= 0;
    iob^.request.message.node.pri:= 20;

    IF (channel = -1) THEN
      iob^.data:= ADR(anychan[0]);
      iob^.length:= 4;
 ELSIF ((channel >= 0) AND (channel < maxChan)) THEN
      IF (usertask[channel] # NIL) THEN
        RETURN (notYourChannel); END;

      iob^.data:= ADR(anychan[channel]);
      iob^.length:= 1;

    ELSE
      RETURN (badChannelSelected);
    END;

    iob^.request.command:= allocate;
    iob^.request.flags:= noWait + quick;
    BeginIO(iob);
    WaitIO(iob);
    error:= LONGINT(iob^.request.error);
    IF (error # 0) THEN RETURN error; END;

    CASE LONGINT(iob^.request.unit) OF
      1 : channum:= 0; |
      2 : channum:= 1; |
      4 : channum:= 2; |
      8 : channum:= 3;
    ELSE
      RETURN (badChannelSelected);
    END;

    unit[channum]:= iob^.request.unit;
    key[channum]:= iob^.allocKey;
    usertask[channum]:= FindTask(0);

    RETURN channum;
END GetChannel;

(*-----------------------------------------------*)
(* IsThatMyChan determina se possediamo ancora un
determinato canale. L'audio device è fatto in modo
da permettere che venga fatta una richiesta a
priorità più alta per un canale già occupato da una
richiesta a priorità più bassa. La richiesta a
priorità più alta può rubare il canale all'altra.
Questa caratteristica potrebbe essere implementata
nella prossima versione di audiotools, nella quale,
in dipendenza dalla priorità del task che sta
girando, un task con priorità più alta potrebbe
avere successo nell'usare GetChannel per un canale
già occupato.
*)

PROCEDURE IsThatMyChan(channel: LONGINT): LONGINT;
 BEGIN
    IF ((channel < 0) OR (channel > maxChan-1)) THEN
      RETURN (badChannelSelected);
    ELSIF (usertask[channel] # FindTask(0)) THEN
      RETURN (notYourChannel); END;
    RETURN 0;
END IsThatMyChan;

(*--------------Procedure interne----------------*)

PROCEDURE ControlChannel(channel: LONGINT;
                         command: CARDINAL): LONGINT;
 VAR
    error: LONGINT;
    iob: ExtIOBPtr;
    controlIOB: ExtIOB;
 BEGIN
    error:= IsThatMyChan(channel);
    IF (error # 0) THEN RETURN error; END;

    iob:= ADR(controlIOB);
    iob^.request.device:= device;
    iob^.request.message.replyPort:= controlPort;
    iob^.request.unit:= unit[channel];
```

```
    iob^.allocKey:= key[channel];

    iob^.request.command:= command;
    IF (command = free) THEN
      iob^.request.flags:= noWait + quick;
    ELSE
      iob^.request.flags:= quick;
    END;

    BeginIO(iob);
    WaitIO(iob);
    error:= LONGINT(iob^.request.error);
    RETURN error;
END ControlChannel;

(*---------Procedure di supporto-----------------*)

PROCEDURE StartChannel(channel: LONGINT): LONGINT;
 BEGIN
   RETURN ControlChannel(channel, start);
END StartChannel;

PROCEDURE StopChannel(channel: LONGINT): LONGINT;
 BEGIN
   RETURN ControlChannel(channel, stop);
END StopChannel;

PROCEDURE ResetChannel(channel: LONGINT): LONGINT;
 BEGIN
   RETURN ControlChannel(channel, reset);
END ResetChannel;

PROCEDURE FlushChannel(channel: LONGINT): LONGINT;
 BEGIN
   RETURN ControlChannel(channel, flush);
END FlushChannel;

PROCEDURE FreeChannel(channel: LONGINT): LONGINT;
 VAR
   error: LONGINT;
 BEGIN
   error:= ControlChannel(channel, free);
   IF (error # 0) THEN RETURN error; END;
   usertask[channel]:= NIL;
   RETURN 0;
END FreeChannel;

(*-----------------------------------------------*)
(* CheckIfDone - per vedere se tutto è finito PRIMA
di chiamare FinishAudio.
*)

PROCEDURE CheckIfDone(): BOOLEAN;
 BEGIN
   RETURN CheckIOBDone();
END CheckIfDone;

(*-----------------------------------------------*)
(* SetPV - stabilisce il periodo e il volume della
nota che sta suonando.
*)

PROCEDURE SetPV(channel: LONGINT;
                period, volume: CARDINAL): LONGINT;
 VAR
   error: LONGINT
   iob: ExtIOBPtr;
   controlIOB: ExtIOB;
 BEGIN
   error:= IsThatMyChan(channel);
   IF (error # 0) THEN RETURN error; END;

   iob:= ADR(controlIOB);
   iob^.request.device:= device;
   iob^.request.message.replyPort:= controlPort;
   iob^.request.unit:= unit[channel];
   iob^.allocKey:= key[channel];

   iob^.period:= period;
   iob^.volume:= volume;

   iob^.request.command:= perVol;
   iob^.request.flags:= quick + pervol;
   BeginIO(iob);
   WaitIO(iob);
   error:= LONGINT(iob^.request.error);
   RETURN error;
END SetPV;

(*-----------------------------------------------*)
(* SetWave - alloca della CHIP RAM se necessario
(solo una volta per canale) e vi copia (espandendo
l'onda) l'ARRAY [0..255] OF BYTE dell'utente, dove
ogni elemento dell'ARRAY deve essere compreso fra
-128 e 127, dal momento che il DMA audio legge una
word (16 bit) alla volta ed esamina due byte.  *)

PROCEDURE SetWave(channel: LONGINT;
            VAR waveform: ARRAY OF BYTE): LONGINT;
 VAR
   error: LONGINT;
   i, j, rate: CARDINAL;
   tmptr: ADDRESS;
 BEGIN
   error:= IsThatMyChan(channel);
   IF (error # 0) THEN RETURN error; END;

   IF (chipaudio[channel] # 0) THEN
     IF (datalength[channel] # waveSize) THEN
       FreeMem(chipaudio[channel],
                              datalength[channel]);
       chipaudio[channel]:= 0;
       datalength[channel]:= 0;
     END;
   END;

   IF (chipaudio[channel] = 0) THEN
     chipaudio[channel]:= AllocMem(waveSize,
                          MemReqSet{chip, memClear});
     IF (chipaudio[channel] = 0) THEN
       RETURN (outOfMemory); END;
     datalength[channel]:= waveSize;
   END;
```

```
tmptr:= chipaudio[channel];
   rate:= 1;
   FOR i:= 0 TO 8 BY 1 DO
     j:= 0;
     REPEAT
       tmptr^:=waveform[j]; INC(tmptr);
       j:= j + rate;
     UNTIL j > 255;
     rate:= rate * 2;
   END;
   RETURN 0;
END SetWave;

(*-----------------------------------------------*)
(* SetSamp - alloca della CHIP RAM se necessario
(solo una volta per canale), a meno che "length" =
0, nel qual caso libera la CHIP RAM utilizzata.  Se
"length" è diversa da zero, copia byte per byte dal
buffer "sampleaudio" alla CHIP RAM, a meno che
"sampleaudio" non sia uguale a zero, nel qual caso
alloca semplicemente la CHIP RAM senza copiarvi
niente e restituisce una nuova "sampleaudio"
all'utente (cosa utile se gli esempi sono caricati
da disco). Notate che ogni elemento di "sampleaudio"
deve essere compreso fra - 128 e 127 dal momento che
il DMA audio legge una word (16 bit) alla volta ed
esamina due byte.  *)

PROCEDURE SetSamp(channel: LONGINT;
             VAR sampleaudio: ADDRESS;
                 length: LONGINT): LONGINT;
 VAR
   error: LONGINT;
   j: LONGINT;
   chiptr, samptr: ADDRESS;
 BEGIN
   error:= IsThatMyChan(channel);
   IF (error # 0) THEN RETURN error; END;

  IF (chipaudio[channel] # 0) THEN
   FreeMem(chipaudio[channel], datalength[channel]);
   chipaudio[channel]:= 0; datalength[channel]:= 0;
  END;

   IF (length = 0) THEN RETURN 0; END;
   IF (length > 131072) THEN length:= 131072; END;

   IF (chipaudio[channel] = 0) THEN
     chipaudio[channel]:= AllocMem(length,
                       MemReqSet{chip, memClear});
     IF (chipaudio[channel] = 0) THEN
       RETURN (outOfMemory); END;
     datalength[channel]:= length;
   END;

   IF (sampleaudio = 0) THEN
     sampleaudio:= chipaudio[channel];
     RETURN 0;
   END;

   chiptr:= chipaudio[channel];
   samptr:= sampleaudio;
   FOR j:= 1 TO length BY 1 DO
     chiptr^:= samptr^; INC(chiptr); INC(samptr);
   END;
   RETURN 0;
END SetSamp;

(*--------------Procedure interne----------------*)

PROCEDURE PlayXXXX(channel: LONGINT;
                   wfptr:   ADDRESS;
                   len:     LONGCARD;
                   per:     CARDINAL;
                   vol:     CARDINAL;
                   cycles:  CARDINAL;
                   priority:    Byte;
                   messageport: MsgPortPtr;
                   id:      LONGINT): LONGINT;
VAR
   error: LONGINT;
   iob: ExtIOBPtr;
 BEGIN
   iob:= GetIOB(channel);
   IF (iob # NIL) THEN
     iob^.request.unit:= unit[channel];
     iob^.allocKey:= key[channel];
     iob^.data:= wfptr;
     iob^.length:= len;
     iob^.period:= per;
     iob^.volume:= vol;
     iob^.cycles:= cycles;
     iob^.request.message.node.pri:= priority;
     iob^.identifier:= id;
     iob^.request.command:= write;
     iob^.request.flags:= pervol;

     iob^.writeMsg.replyPort:= messageport;
     IF (messageport # NIL) THEN
       iob^.request.flags:= iob^.request.flags +
                                    writeMessage;
     END;
     BeginIO(iob);
     RETURN 0;
   END;
   RETURN (outOfMemory);
END PlayXXXX;

(*-----------------------------------------------*)
(* PlayNote - fa partire un suono sul canale
specificato. Questa piccola ma utile routine prende
una nota e la suona sul canale richiesto. La nota è
rappresentata da un intero compreso fra 0 e 11 più
un multiplo di 12 per le ottave sopra la prima, il
che porta a una scala di note comprese fra 0 e 95.
La forma d'onda utilizzata è determinata aggiungendo
un indice (woffset[]) che dipende dall'ottava alla
forma d'onda di chipaudio[channel], così come
preparata da SetWave. la lunghezza della forma
d'onda (wlen[]) dipende anch'essa dall'ottava.
Notate che la numerazione delle ottave parte da
zero, non da uno. Il periodo e il volume possono
```

```
essere modificati successivamente usando SetPV.
*)

PROCEDURE PlayNote(channel: LONGINT;
                   note:    CARDINAL;
                   volume:  CARDINAL;
                   duration: CARDINAL;
                   priority: Byte;
                   messageport: MsgPortPtr;
                   id: LONGINT);
VAR
   error: LONGINT;
   period, octave: CARDINAL;
   ipart, jpart: CARDINAL;
   length: LONGCARD;
   wavepointer: ADDRESS;
   cycles: CARDINAL;
 BEGIN
   error:= IsThatMyChan(channel);
   IF (error # 0) THEN RETURN; END;

   IF (note > 95) THEN note:= 95; END;
   IF (volume > 64) THEN volume:= 64; END;
   octave:= note DIV 12;

   IF (chipaudio[channel] = 0) THEN RETURN; END;
   wavepointer:= chipaudio[channel] +
                         ADDRESS(woffsets[octave]);
   length:= wlen[octave];
   period:= perval[note MOD 12];

   IF (duration > 1000) THEN
     ipart:= duration DIV 1000; ELSE ipart:= 0; END;
   jpart:= duration - (ipart * 1000);

   cycles:= ((LONGCARD(audClock) * ipart)+
            (LONGCARD(audClock) * jpart) DIV 1000)
           DIV (LONGCARD(length) * period);

   IF ((cycles = 0) AND (duration # 0)) THEN
     cycles:= 1; END;

   error:= PlayXXXX(channel, wavepointer, length,
                    period, volume, cycles,
                    priority, messageport, id);
   RETURN
END PlayNote;

(*-----------------------------------------------*)
(* PlayFreq - in questa versione funziona solo con
valori o frequenze scalari. Il valore minimo è 28Hz,
il massimo in pratica è 7000Hz. Il periodo viene
calcolato dalla frequenza e deve essere compreso fra
127 e 500 altrimenti se la frequenza eccede quelle
che abbiamo nella nostra tabella, dobbiamo rifiutare
di eseguire il comando.
*)

PROCEDURE PlayFreq(channel: LONGINT;
                   freq:    CARDINAL;
                   volume:  CARDINAL;
                   duration: CARDINAL;
                   priority: Byte;
                   messageport: MsgPortPtr;
                   id: LONGINT);
VAR
   error: LONGINT;
   period, octave: CARDINAL;
   ipart, jpart: CARDINAL;
   length: LONGCARD;
   wavepointer: ADDRESS;
   cycles: CARDINAL;
   i: CARDINAL;
   accept: BOOLEAN;
 BEGIN
   error:= IsThatMyChan(channel);
   IF (error # 0) THEN RETURN; END;

   IF (freq = 0) THEN RETURN; END;
   IF (volume > 64) THEN volume:= 64; END;

   i:= 0;
   LOOP
     octave:= i;
     accept:= FALSE;
     period:= LONGCARD(audClock) DIV
                 (LONGCARD(freq) * (wlen[octave]));
     IF (period > 500) THEN EXIT; END;
     IF (period > 127) THEN
       accept:= TRUE; EXIT; END;
     i:= i+1; IF (i > 8) THEN EXIT; END;
   END;
   IF (accept = FALSE) THEN RETURN; END;

   IF (chipaudio[channel] = 0) THEN RETURN; END;
   wavepointer:= chipaudio[channel] +
                         ADDRESS(woffsets[octave]);
   length:= wlen[octave];

   IF (duration > 1000) THEN
     ipart:= duration DIV 1000; ELSE ipart:= 0; END;
   jpart:= duration - (ipart * 1000);

   cycles:= (LONGCARD(freq) * ipart) +
                 (LONGCARD(freq) * jpart) DIV 1000;
   IF ((cycles = 0) AND (duration # 0)) THEN
     cycles:= 1; END;

   error:= PlayXXXX(channel, wavepointer, length,
                    period, volume, cycles,
                    priority, messageport, id);
   RETURN
END PlayFreq;

(*-----------------------------------------------*)
(* MayGetNote - viene utilizzata per sincronizzare
le routine audio Play utilizzandone i parametri
"messageport" e "id". Il parametro "uport" è il
puntatore alla porta messaggi che abbiamo ricevuto
da InitAudio. Quando flag = FALSE la routine esce
immediatamente con id = 0 (nessuna id disponibile) o
con il valore della prima id che arriva alla porta.
```

```
Quando flag = TRUE la routine aspetta se (e solo se)
non c'è id. In altre parole, potete costringere il
vostro task a dormire fino a quando non inizia
l'esecuzione della prossima nota. Potete decidere
cosa fare per ogni singola nota.  ATTENZIONE - se
non ci sono più note con "messageport" diversa zero
nella catena e voi specificate TRUE per il parametro
"flag", il vostro task potrebbe dormire
all'infinito!!
*)

PROCEDURE MayGetNote(uport: MsgPortPtr;
                     flag:  BOOLEAN): LONGINT;
 VAR
   aum: auMsgPtr;
 BEGIN
   LOOP
     aum:= auMsgPtr(GetMsg(uport));

     IF (aum # NIL) THEN

     (* L'utente ha visto questo messaggio quindi il
     sistema può deallocare la strutture iob che lo
     contengono che verranno ricevute. Adesso che
     abbiamo ricevuto il messaggio alla nostra reply
     port possiamo farne quello che vogliamo perché
     ci appartiene. Settiamo la variabile replyport
     a zero per segnalare a FreeIOB che può
     veramente fare il suo lavoro!
     *)

       aum^.message.replyPort:= NIL;
       EXIT;
     END;

     IF (flag = TRUE) THEN
       WaitPort(uport);
       flag:= FALSE;
     END;
   END;
   RETURN (aum^.identifier);
END MayGetNote;

(*-----------------------------------------------*)
(* PlaySamp - suona un pezzo campionato. E' identico
a PlayFreq ma i parametri vengono interpretati
diversamente. La variabile "freq" diventa "period"
intesa come frequenza di campionamento e deve essere
compresa fra 127 e 500. La variabile "duration" è
ancora espressa in millesimi di secondo (come accade
con lo stesso audio device, quando questa variabile
contiene zero la nota viene suonata all'infinito,
fino a quando l'audio device non viene resettato o
fino a quando il canale non è liberato con FLUSH o
fino a quando questo comando non viene
esplicitamente interrotto.
*)

PROCEDURE PlaySamp(channel: LONGINT;
                   period:  CARDINAL;
                   volume:  CARDINAL;
                   duration: CARDINAL;
                   priority: Byte;
                   messageport: MsgPortPtr;
                   id: LONGINT);
VAR
   error: LONGINT;
   wavepointer: ADDRESS;
   cycles: CARDINAL;
   ipart, jpart: CARDINAL;
   length: LONGCARD;
 BEGIN
   error:= IsThatMyChan(channel);
   IF (error # 0) THEN RETURN; END;

   IF (period > 500) THEN period:= 500;
   ELSIF (period < 127) THEN period:= 127; END;

   IF (volume > 64) THEN volume:= 64; END;

   IF (chipaudio[channel] = 0) THEN RETURN; END;
   wavepointer:= chipaudio[channel];
   length:= datalength[channel];

   IF (duration > 1000) THEN
     ipart:= duration DIV 1000; ELSE ipart:= 0; END;
   jpart:= duration - (ipart * 1000);

   cycles:= ((LONGCARD(audClock) * ipart) +
             (LONGCARD(audClock) * jpart) DIV 1000)
            DIV (LONGCARD(length) * period);
   IF ((cycles = 0) AND (duration # 0)) THEN
     cycles:= 1; END;

   error:= PlayXXXX(channel, wavepointer, length,
                    period, volume, cycles,
                    priority, messageport, id);
   RETURN
END PlaySamp;

(*-----------------------------------------------*)
(* FinishAudio - Se l'utente dice FinishAudio, vuole
dire FERMA L'AUDIO.  Rimuovi qualsiasi cosa stia
attualmente suonando, ADESSO. Potete usare
"CheckIfDone()" per vedere se tutto è terminato,
PRIMA di chiamare FinishAudio. Se CheckIfDone()
restituisce FALSE vuole dire che qualcosa sta ancora
suonando.
*)

PROCEDURE FinishAudio(uport: MsgPortPtr);
 VAR
   error: LONGINT;
   aum: auMsgPtr;
   i: LONGINT;

 BEGIN
   IF (uport # NIL) THEN
     FOR i:= 0 TO maxChan-1 BY 1 DO
       error:= FlushChannel(i);
     END;
```

```
      WHILE (CheckIOBDone() = FALSE) DO
        Delay(12);
      END;

      aum:= auMsgPtr(GetMsg(uport));
      WHILE (aum # NIL) DO
        aum^.message.replyPort:= NIL;
        aum:= auMsgPtr(GetMsg(uport));
      END;

      ReEmployIOB();
FOR i:= 0 TO maxChan-1 BY 1 DO
        error:= FreeChannel(i);
      END;
      DeletePort(uport);
    END;

    IF (device # NIL) THEN
      CloseDevice(ADR(openIOB)); END;
    FOR i:= 0 TO maxChan-1 BY 1 DO
      IF (chipaudio[i] # 0) THEN
        FreeMem(chipaudio[i], datalength[i]);
        chipaudio[i]:= 0; datalength[i]:= 0;
      END;
      IF (replyPort[i] # NIL) THEN
        DeletePort(replyPort[i]); END;
    END;
    IF (controlPort # NIL) THEN
      DeletePort(controlPort); END;

END FinishAudio;

END AudioTools.imp
```

Listato 3:

```
MODULE AudioDemo;

FROM SYSTEM IMPORT
 ADR, ADDRESS, BYTE, FFP;

FROM AudioTools IMPORT
 InitAudio, GetChannel, StopChannel, StartChannel,
 SetPV, SetWave, SetSamp, PlayNote, PlayFreq,
 MayGetNote, PlaySamp, CheckIfDone, FinishAudio;

FROM Arts IMPORT TermProcedure;

FROM Dos IMPORT Delay;

FROM Exec IMPORT Byte, UByte, MsgPortPtr;

FROM RandomNumber IMPORT RND;

FROM Terminal IMPORT WriteLn, WriteString;

TYPE
  waveArray = ARRAY [0..255] OF BYTE;
CONST
  noiselen = 1024;
VAR
  i: CARDINAL;
  channel, error, myid: LONGINT;
  myport: MsgPortPtr;
  sawtooth: waveArray;
  triangle: waveArray;
  square:   waveArray;
  sinewave: waveArray;
  noise: ARRAY [0..noiselen-1] OF BYTE;
  noiseptr: ADDRESS;

PROCEDURE MakeWaveArrays;
 VAR
   i: INTEGER;
   trv, sqv: Byte;
   sine, cosine, step: FFP;
 BEGIN
   trv:= 0; sqv:= 127;
   sine:= 0.0; cosine:= 1.0; step:= 0.0247;
   FOR i:= 0 TO 255 BY 1 DO
     sawtooth[i]:= Byte(127 - i);
     triangle[i]:= Byte(trv);
     square[i]:=   Byte(sqv);
     sinewave[i]:= Byte(INTEGER(127.0 * sine));
     IF (i > 62) AND (i < 190) THEN
       DEC(trv, 2); ELSE INC(trv, 2); END;
     IF (i > 127) THEN sqv:= -128; END;
     cosine:= cosine - (step * sine);
     sine:= sine + (step * cosine);
   END;
END MakeWaveArrays;

PROCEDURE MakeNoiseArray;
 VAR
   i: CARDINAL;
 BEGIN
   FOR i:= 0 TO noiselen-1 BY 1 DO
     noise[i]:= Byte(127 - RND(256));
   END;
   noiseptr:= ADR(noise[0]);
END MakeNoiseArray;

PROCEDURE Cleanup;
 BEGIN
   IF (myport # NIL) THEN
     FinishAudio(myport);
     WriteString("Done!");
     WriteLn;
   END;
END Cleanup;

BEGIN  (* AudioDemo *)
  TermProcedure(Cleanup);
  WriteLn;
  WriteString("Demo di AudioTools. ");
  WriteLn;
  MakeWaveArrays;
  WriteLn;
  myport:= InitAudio();
  IF (myport = NIL) THEN
    WriteString("Problemi con InitAudio!"); RETURN;
  END;
```

```
FOR i:= 0 TO 3 BY 1 DO
    channel:= GetChannel(-1);
    IF (channel = 1) THEN
      WriteString("Non posso accedere al canale!");
      RETURN;
    END;
    error:= StopChannel(channel);
    IF (error # 0) THEN
      WriteString("Errore nel fermare un canale!");
      RETURN;
    END;
  END;

  WriteString("Allocati e fermati tutti i canali");
  WriteLn;

  error:= SetWave(0, sawtooth);
  IF (error # 0) THEN
    WriteString("Errore durante SetWave, canale 0");
    RETURN;
  END;
  error:= SetWave(1, triangle);
  IF (error # 0) THEN
    WriteString("Errore durante SetWave, canale 1");
    RETURN;
  END;
  error:= SetWave(2, square);
  IF (error # 0) THEN
    WriteString("Errore durante SetWave, canale 2");
    RETURN;
  END;
  error:= SetWave(3, sinewave);
  IF (error # 0) THEN
    WriteString("Errore durante SetWave, canale 3");
    RETURN;
  END;
  WriteString("SetWave per le 4 forme d'onda");
  WriteLn;

  FOR i:= 0 TO 48 BY 1 DO
    PlayFreq(0, 10*i, 32, 125, 0, NIL, 0);
    PlayNote(1, 95-i, 32, 125, 0, NIL, 0);
  END;
  WriteString("PlayNotes e PlayFreqs basse freq.");
  WriteLn;

  PlayNote(0, 3, 63, 500, 0, myport, 7);
  WriteString("Test messaggio con id = 7");
  WriteLn;

  FOR i:= 50 TO 95 BY 1 DO
    PlayFreq(0, 10*i, 32, 125, 0, NIL, 0);
    PlayNote(1, 95-i, 32, 125, 0, NIL, 0);
  END;
  WriteString("PlayNotes e PlayFreqs alte freq.");
  WriteLn;

  FOR i:= 0 TO 3 BY 1 DO
    error:= StartChannel(i);
    IF (error # 0) THEN
      WriteString("Errore durante StartChannel!");
      RETURN;
    END;
  END;
  WriteString("Tutti i canali O.K.");
  WriteLn; WriteLn;

  WriteString("DORMIAMO - myid ci sveglierà.");
  WriteLn;
  myid:= MayGetNote(myport, TRUE);
  WriteLn;
  WriteString("Ho identificato una nota come ");
  IF (myid = 7) THEN
    WriteString("7"); ELSE WriteString("0"); END;
  WriteLn; WriteLn;

  WriteString("Aspettiamo 6 secondi...");
  Delay(300);
  WriteString("CheckIfDone ? - ");
  IF (CheckIfDone() = FALSE) THEN
    WriteString("FALSE");
  ELSE WriteString("TRUE"); END;
  WriteLn; WriteLn;
  WriteString("PlayNotes con durata > 2000).");
  WriteLn;
  PlayNote(0, 23, 32, 2000, 0, NIL, 0);
  PlayNote(1, 27, 32, 2300, 0, NIL, 0);
  PlayNote(2, 30, 32, 2600, 0, NIL, 0);
  PlayNote(3, 35, 32, 2900, 0, NIL, 0);
  WriteLn;
  Delay(150);
  WriteString("Dovremmo aver finito ultima nota.");
  WriteString("CheckIfDone ? - ");
  IF CheckIfDone() THEN WriteString("TRUE");
  ELSE WriteString("FALSE"); END;
  WriteLn; WriteLn;

  error:= SetSamp(0, noiseptr, noiselen);
  IF (error # 0) THEN
    WriteString("Errore durante SetSamp, canale 0");
    RETURN;
  END;
  WriteString("Un pò di rumore usando PlaySamp");
  WriteLn;
  PlaySamp(0, 327, 48, 6000, 0, NIL, 0);

  FOR i:= 6 TO 1 BY -1 DO
    Delay(50);
    error:= SetPV(0, 327, i*8);
    IF (error # 0) THEN
      WriteString("Errore durante SetPV, canale 0");
      RETURN;
    END;
  END;

  WriteString("Il rumore dovrebbe essere finito.");
  WriteLn; WriteLn;
  WriteString("Pronto a finire il demo audio.");
  WriteLn;
  RETURN;
END AudioDemo.mod
```

Jackson

SOFTWARE & RIVISTE

D.O.C.

Scegli i giochi più freschi, quelli appena sfornati dal mercato internazionale. Poi, in edicola, chiedi le riviste più ricche di informazioni e idee per utilizzare il tuo computer nel modo più O.K. Ora unisci al tutto qualità, esperienza, tanta serietà, una lunga tradizione di fiducia e un pizzico di divertimento. Ecco fatto. Hai trovato il marchio di qualità che solo Jackson ti può offrire: è il marchio Jackson Software D.O.C. D'ora in poi cercalo sempre.

GRUPPO EDITORIALE JACKSON

AREA CONSUMER

Scegli il meglio: scegli Jackson.

Compila e spedisci il coupon a: **GRUPPO EDITORIALE JACKSON AREA CONSUMER via Rosellini, 12 - 20123 MILANO**

Riceverai gratuitamente il nuovissimo adesivo "Jackson D.O.C."

NOME _____ COGNOME _____

INDIRIZZO _____ TEL. ( ) _____

CAP. _____ CITTÀ _____ PROV. _____

ETÀ _____ PROFESSIONE _____

QUALE COMPUTER POSSIEDI? _____

QUALI RIVISTE LEGGI? DI INFORMATICA _____

_____ ALTRE _____

_____ SEI ABBONATO AD UNA RIVISTA JACKSON? SI ☐ NO ☐

QUALE? _____

_____ SEI TITOLARE JACKSON CARD? SI ☐ NO ☐

SILVER CARD N° _____ GOLD CARD N°. _____

PER I TUOI ACQUISTI RIVOLGITI AI PUNTI VENDITA UFFICIALI COMMODORE

LOMBARDIA

MILANO ALPHA COMPUTER Via Tavazzano 14 • AL RISPARMIO V.le Monza 204 • BCS Via Mantegani 11 • BRAHA ALBERTO Via P. Capponi, 5 • E.D.S. C.so Porta Ticinese, 4 • E.S.C. Via Roggia Scagna, 7 • FAREF Via A. Volta, 21 • FLOPPERIA SRL V.le Montenero, 31 • GIGLIONI LAURA Via D'Ovidio, 8 • GIGLIONI V.le Luigi Sturzo, 45 • LOGITEK Via Golgi, 60 • MARCUCCI Via Fratelli Bronzetti, 37 • MELCHIONI Via P. Colletta, 37 • MESSAGGERIE MUSICALI Galleria Del Corso, 2 • NEWEL Via Mac Mahon, 75 • RIVOLA Via Vitruvio, 43 • **S.ANGELO LODIGIANO** FERRARI LUIGI Via Madre Cabrini, 44 • **BARLASSINA** F.LLI GALIMBERTI Via Nazionale dei Giovi, 28/36 • **BOVISIO MASCIAGO** R & C C.so Milano, 118 •**CINISELLO BALSAMO** GBC V.le Matteotti, 66 • **MILANO** GBC Via Petrella, 6 • **MILANO** GBC Via Cantoni, 7 •**COLOGNO MONZESE** CASA DELLA MUSICA Via Indipendenza, 21 • **CORBETTA** PENATI Via Verdi, 28/30 • **CORSICO** EPM System V.le Italia,12 • **DESIO** P.GIORGIO OSTELLARI Via Milano, 300 •**LEGNANO** CENTROCOMPUTER PANDOLFI Via Corridoni,18 •**LISSONE** COMPUTEAM Via Vecellio, 41 • **LODI** FUTURA Via Solferino, 31 •**LODI** M.B.M. C.so Roma, 112 •**MELEGNANO** L'AMICO DEL COMPUTER SAS V.le Lombardia, 17 • **MONZA** BIT 84 Via Italia, 4 • **NOVATE MILANESE** IL CURSORE Via Cavour, 35 • **OPERA** I.C.O. Via dei Tigli, 14 • **SESTO SAN GIOVANNI** NIWA HARD & SOFT Via Bruno Buozzi, 94 • **VIMODRONE BERGAMO** IL COMPUTER SERVICE SHOP Via Padana Superiore, 197 • **BERGAMO** COMIF Via Autolinee, 10 • CORDANI Via dei Caniana, 8 • D.R.B. Via Borgo Palazzo, 65 • NEW SYSTEMS POINT Via Paglia, 36 • **ALZANO LOMBARDO** BERTULEZZI GIOVANNI Via Fantoni, 48 •**CARVICO** COMPUTER TEAM hi-tec Via Verdi, 1/B •**LOVERE** OTTICO OPTOMETRISTA ROVETTA P.zza Garibaldi, 6 • **SAN PELLEGRINO TERME** A.I.S. INTERNATIONAL Via S. Carlo, 25 •**SARNICO** COMPUTER POINT Via Lantieri, 52 •**BRESCIA** SISTHEMA Via Roma, 45 •**URGNANO BRESCIA E PROVINCIA** A.B.INFORMATICA Strada Statale Cremasca, 66 •**BRESCIA** COMPUTER CENTER Via Cipro, 62 • INFORMATICA 2000 Via Stazione, 16/B •MASTER INFORMATICA Via F.lli Ugoni, 10/8 •VIGASIO MARIO Portici Zanardelli, 3 •**BRENO** MISTER BIT Via Mazzini, 70 • **CASTREZZATO** CAVALLI PIETRO Via 10 Giornate, 14B •**CHIARI** (BS) VIETTI GIUSEPPE Via Milano, 1/B •**DESENZANO DEL GARDA** MEGABYTE P.zza Maluezzi, 1 •**GHEDI** DITTA BARESI RINO & C. Via XX Settembre, 7 •**GRATACASOLO** INFO CAM Via Provinciale, 3 •**COMO** IL COMPUTER Via Indipendenza, 90 •2M ELETTRONICA Via Sacco, 3 • **BARZANO'** ELTRONGROS VIA L. Da Vinci, 54 •**CASSAGO BRIANZA** EGA Via Mazzini, 42 • **ERBA** DATA FOUND computer shop Via A. Volta, 4 •**LECCO** CIMA ELETTRONICA Via Leonardo Da Vinci, 7 •FUMAGALLI Via Cairoli, 48 • **OLGIATE COMASCO** RIGHI ELETTRONICA Via G. Leopardi, 26 •**CREMONA** MONDO COMPUTER Via Giuseppina, 11/B • PRISMA Via Buoso da Novara, 8 • TELCO P.zza Marconi, 2/A • **CREMA** ELCOM/GBC Via VI Novembre, 56/58 • EUROELETTRONICA Via XX Settembre, 92/A • **MANTOVA E PROVINCIA** COMPUTER SAS Galleria, 7 •32 BIT (Computer Studio) Via Cesare Battisti, 14 • ELETTRONICA di BASSO V.le Risorgimento, 69 •**PAVIA** POLIWARE SRL C.so Carlo Alberto, 76 • **VIGEVANO** LOGICA INFORMATICA V.le Monte Grappa, 34 •M. VISENTIN C.so Vittorio Emanuele, 76 •**SONDRIO** CIPOLLA MAURO Via Tremogge, 25 • **SAN PIETRO DI BERBENNO** FOTONOVA V.Valeriana, 1 • **VARESE** DIMECO SISTEMI Via Garibaldi •IL CENTRO ELETTRONICO Via Morazzone, 2 • SUPERGAMES Via Carrobbio, 13 •**BUSTO ARSIZIO** BIT Via Gavinana, 17 • MASTER PIX SNC Via S. Michele, 3 •**CASTELLANZA** CRESPI GIUSEPPE & C. V.le Lombardia, 59 •**GALLARATE** PUNTO UFFICIO Via Raffaello Sanzio, 8 •GEMONIO SIDALCO Via Caprera, 10 •**GERENZANO** (VA) GRANDI MAGAZZINI BOSSI Via Clerici, 196 •**SESTO CALENDE** (VA) J.A.C. nuove tecnologiè Via Matteotti, 38

PIEMONTE

ALESSANDRIA BIT MICRO Via Mazzini, 102 •SERVIZI INFORMATICI Via Alessandro III, 53 •**TORTONA** S.G.E. ELETTRONICA Via Bandello, 19 •**ASTI** RECORD C.so Alfieri, 166/3 • **ASTI CUNEO** ROSSI COMPUTERS C.so Nizza, 42 •**CUNEO** STUDIO SOFTWARE C.so Nizza, 4 •PUNTO BIT C.so Langhe, 26/C •ALBA SDI Via Vittorio Emanuele, 250 BRA • **FOSSANO** ASCHIERI GIANFRANCO C.so Emanuele Filiberto, 6 • BOSETTI Via Roma, 149 • **NOVARA** ELCOM SRL C.so Mazzini, 11 •PROGRAMMA 3 V.le Buonarroti,8 •PUNTO VIDEO C.so Risorgimento, 39/8 • **ARONA** COMPUTER Via Monte Zeda, 4 • **BORGOMANERO** ALL COMPUTER C.so Garibaldi,106 • **DOMODOSSOLA** MICROLOGIC Via Giovanni III, 2 • INTRA ELLIOTT COMPUTER SHOP Via Don Minzoni, 32 • **TORINO** ABA ELETTRONICA Via C. Fossati, 5/P • ALEX COMPUTER E GIOCHI C.so Francia, 333/4 • **VILLASTELLONE** CDM ELETTRONICA Via Marocchetti, 17 spedizione: C.so Savona, 43 • **TORINO** COMPUTER HOME SNC Via San Donato, 46/D • COMPUTING NEW Via Marco Polo, 40/E • DE BUG C.so Vittorio Emanuele II, 22 • DESME UNIVERSAL Via San Secondo, 95 • F.D.S. ALTERIO Via Borgaro, 86/D • IL COMPUTER Via Nicola Fabrizi, 140 bis • INFORMATICA ITALIA C.so Re Umberto, 129 (128) • MT INFORMATICA C.so Giulio Cesare, 58 • MUSIC 'S SHOP C.so Potenza, 177 • NEWBUSINESS COMPUTER Via Nizza, 45/F • PLAYGAMES SHOP Via Carlo Alberto, 39/E • RADIO TV MIRAFIORI C.so Unione Sovietica, 381 • SMT ELETTRONICA Via Bibiana, 83/B • TELERITZ C.so Traiano, 34 • **CHIERI** PAUL E CHICO VIDEOSOUND Via Vittorio Emanuele, 52 • **CIRIE'** BIT INFORMATICA Via Vittorio Emanuele, 154 • **COLLEGNO** HI-FI CLUB C.so Francia, 92C • **FAVRIA** MISTER PERSONAL Via Cattaneo, 52 • **IVREA** C.S.S. Stradale Torino N. 73 • MONCALIERI BAS C.so Roma, 47 •**PINEROLO** CERUTTI MAURO C.so Torino, 234 • **RIVAROLO** EUREX C.so Indipendenza, 5 •**RIVOLI** C.SE DIAM INFORMATICA C.so Francia, 1 46/bis • FULLINFORMATICA Via Vittorio Veneto, 25 • **SAN MAURO TORINESE** PANORAMA TORINO SPA Strada Settimo, 371 • **SETTIMO TORINESE** GAMMA COMPUTER Via Cavour, 3 A/B •**VERCELLI E PROVINCIA** DITTA ELETTROGAMMA C.so Bormida, 27 • ELETTRONICA Strada Torino, 15 • **BIELLA** C.S.I. TEOREMA Via Losana, 9 • SIGEST SRL Via Bertodano, 8 • **BORGOSESIA** REMONDINO FRANCO Via Roma, 5 • **COSSATO** FOTOSTUDIO TREVISAN Via XXV Aprile, 24/B • **TRINO** STUDIO FOTOGRAFICO IMARISIO P.zza Martiri Libertà, 7

VENETO

BELLUNO UP TO DATE di VIEL RENZO Via Vittorio Veneto, 43 • **PADOVA E PROVINCIA** GUERRA COMPUTERS FELTRE Via Mazzini, 10/C • **PADOVA** BIT SHOP Via Cairoli, 11 • COMPUMANIA Riviera Tisa da Camposampiero, 37 • COMPUTER POINT Via Roma, 63 • D.P.R. Vicolo Lombardo,4 •GIANFRANCO MARCATO Via Madonna della salute, 51/53 spedizione: Via Bergamini, 4 • SARTO COMPUTER Via Armistizio, 79 • ZELLA ADELIO P.zza De Gasperi, 31/A • **CITTADELLA ROVIGO** COMPUTER SERVICE Borgo Treviso, 150 • **ROVIGO TREVISO** CLINICA DEL RASOIO E DEL COMPUTER Via Fiume, 31/33 • **TREVISO** BIT 2000 Via Brandolini D'Adda, 14 • GUERRA EGIDIO &C. V.le Cairoli, 95 • **CONEGLIANO** DE MARIN COMPUTERS Via XX Settembre, 74 • **MONTEBELLUNA** SIDESTREET Via Salvo D'Acquisto, 8 • **PREGANZIOL** FALCONE ELETTROAUDIOVIDEO Via Terraggio, 116 • **VENEZIA E PROVINCIA** TELERADIO Fuga San Marco spedizione: IVANO BERTOLA Via Rossi Rubano • **MESTRE-VENEZIA** TREKILOWATT Via Torre Belfredo, 47 •**CHIOGGIA** T.C.H. Riva Vena, 889 • **SAN DONA' DI PIAVE** (VE) GUERRA COMPUTER Via Vizzotto,29 • REBEL Via F. Crispi, 10 • **SOTTOMARINA** TELFERT di Ferretto Flavio Via Chiesa • **SPINEA** RADIOCESTARO Via Roma, 89 • **VERONA** CASA DELLA RADIO Via Cairoli 10 • TELESAT Via Vasco De Gama ,8 spedizione: Ivano Bertola Via Rossi • **LEGNANO** RUBANO FERRARIN Via dei Massari,10 spedizione: Ivano Bertola Via Rossi • **VICENZA** ELETTRONICA BISELLO V.le Trieste, 427/429 • SCALCHI MARKET Via Cà Balbi, 139 • **CECCATO** GUERRA COMPUTER Via Dell'industrie Alte • **CAVAZZALE** SCHIAVOTTO Via Zanella, 21 • COMPUTER B.COSTO Via del Costo,34

FRIULI VENEZIA GIULIA

GORIZIA THIENE ELETTROCASA Via Roma, 67 • **GORIZIA PORDENONE** E.C.O. ELETTRONICA COMMERCIALE Via F.lli Cossar, 23 • **PORDENONE** RIGO V.le Cossetti, 5 • BRUNO DA PIEVE & C. Via Colombera,I7 • **VILLANOVA DI PRATA TRIESTE** PORCIA MDT P.zza Repubblica, 5 • **TRIESTE** AVANZO GIACOMO P.zza Cavana, 7 • COMPUTER SHOP Via P. Reti • **TRIESTE** COMPUTIGI Via XX. Settembre, 51 • **TRIESTE UDINE** CT Via Pascoli, 4 • **UDINE** MOFERT V.le Europa Unita, 41 spedizione:Mifert,2 Via Leopardi 2 • R.T. SISTEM UDINE Via L. Da Vinci, 99 • **MARTIGNACCO** INDRENO MATTIUSSI & C. Via Liciniana, 58

TRENTINO ALTO ADIGE

BOLZANO COMPUTER POINT Via Roma, 82A • **BOLZANO** MATTEUCCI PRESTIGE Via Museo, 54 spedizione: ELETTRON MATTEUCCI Via Parma • **BRUNICO** RADIO MAIR-ELECTRO Via Centrale 70 • **MERANO** ELECTRO RADIO HENDRICH Via Delle Corse, 106 • **SILANDRO** (BZ) **TRENTO** ELECTRO TAPPEINER P.zza Principale, 90 • **TRENTO** CRONST SAS Via G. Galilei, 25 • ELETTROCASA V.le Verona, 9

LIGURIA

GENOVA ABM COMPUTER P.zza De Ferrari, 24 rosso • **GENOVA SESTRI PONENTE** CENTRO ELETTRONICO Via Chiaravagna, 10R • **GENOVA** COMMERCIALE SOTTORIPA Via Sottoripa, 115/117 • **SAMPIERDARENA GENOVA** COMPUTER VIA D.G. Storace 4/rosso • **GENOVA** FOTOMONDIAL Via Del Campo 3-5-9-11-13 r • LA NASCENTE Via San Luca, 4/1 • **GENOVA IMPERIA** RAPPR-EL Via Borgoratti, 23/R • **IMPERIA CASTELLINO** Via Belgrano, 44 spedizione: SASA COMPUTER • **SANREMO** (Terso) CENTROHI-FI VIDEO Via della Repubblica, 38 • **VENTIMIGLIA CASTELLINO** Via Genova, 48 spedizione: SASA COMPUTER (Terso) • **LA SPEZIA E PROVINCIA** I.L. ELETTRONICA Via Vittorio Veneto, 123 • **FORNOLA DI VEZZANO SAVONA** I.L. ELETTRONICA Via Aurelia, 299 • **SAVONA** ATHENA INFORMATICA Via Carissimo e Crotti, 16/R • **CASTELLINO** C.so Tardy e Benech, 101 spedizione: SASA COMPUTER (Terso)

EMILIA ROMAGNA

PIACENZA COMPUTER LINE Via G. Carducci, 4 • DELTA COMPUTER Via Martiri della Resistenza, 15/G • SOVER Via VI Novembre, 60

TOSCANA

AREZZO DELTA SYSTEM Via Piave, 13 • **FIRENZE** ATEMA Via Benedetto Marcello, 1a-1b • ELETTRONICA CENTOSTELLE Via Cento Stelle, 5/a-b • HELP COMPUTER Via degli Artisti, 15-A • PUNTO SOFT Via Vagnetti, 17 • TELEINFORMATICA TOSCANA Via Bronzino, 36 • **EMPOLI** WAR GAMES SDF Via Raffaello Sanzio, 126/A • **FIGLINE VALDARDO** (FI) NEW E.V.M COMPUTER Via Degli Innocenti, 2 • **PONTASSIEVE** CENTRO INFORMATICA Via Znojmo, 41 • **PRATO** (FI) COSCI F.lli Via Roma, 26 • **GROSSETO** COMPUTER SERVICE Via Dell'Unione, 7 • **LIVORNO** ETA BETAVIA San Francesco, 30 • FUTURA 2 Via Cambini, 19 • **PIOMBINO** ELETTRONICA ALESSI Via Cimarosa, 1 • **LIDO DI CAMAIORE** IL COMPUTER V.le Colombo, 21 6 • S. **ROMANO GARFAGNANA** (LU) SANTI VITTORIO Via Roma, 23 • **MASSA** EURO COMPUTER P.zza B. Bertagnini, 4 • FIRMWARE Galleria Pregliasco, 17 • **CARRARA** (MS) RADIO LUCONI Via Roma, 24/B • **PISA** C.H.S. SNC Via Carlo Cattaneo, 90/92 • ELECTRONIC SERVICE Via Della Vecchia Tranvia, 10 • IT - LAB Via Marche, 8A-8B • ditta TONY HI-FI Via G. Carducci, ang. Canto Dei Nicchio, 2 • **PISTOIA E PROVINCIA** ELECTRONIC SHOP Via Della Madonna, 49 • OFFICE DATA SERVICE Galleria Nazionale, 22 • **MONTECATINI TERME** ZANNI & C. C.so Roma, 45 TEL. 0572-79649 spedizione: Via Forini,l0 • **SIENA** VIDEO MOVIE Via Garibaldi, 17 • **CHIANCIANO TERME** ELECTRIC SHOP Via A. Casini, 51 • **MONTEPULCIANO** ELETTRONICA Via di Gracciano nel Corso, 111

UMBRIA

PERUGIA LA MIGLIORATI Via S. Ercolano, 310 • **BASTIA UMBRA** COMPUTER STUDIO'S VIA IV Novembre, 18/A • **CITTA' DI CASTELLO** WARE Via Dei Casceri, 31 • **MATERA** GIOVANNI GAUDIANO ELECTRONICS Via Roma

PUGLIA

BARI ARTEL Via Guido D'Orso, 9 • COMPUTER'SARTS V.leb Meucci, 12/B • PAULICELLI SABINO & FIGLI Via Fanelli, 231/C • **BARLETTA** F. FAGGELLA C.so Garibaldi, 15 • GIANNI FAGGELLA P.zza D'Aragona, 62A • **CASTELLANA** LONUZZO GIUSEPPE Via Nizza, 21 • **BRINDISI** MARANGI E MICCOLI Via Prov. San Vito, 165 • **FRANCAVILLA FONTANA** MILONE GIOVANNI Via San Francesco D'Assisi, 219 • **FOGGIA** BOTTICELLI GUIDO Via San Pollice, 2 • E.C.I. COMPUTER Via Isonzo, 28 • LA TORRE SAS V.le Michelangelo, 185 • **SAN SEVERO** IL DISCOBOLO Via T. Solis, 15 • **LECCE** BIT Via 95° Reggimento Fanteria, 87/89 • TRICASECEDOK INFORMATICA Via Roma, 31 • **TARANTO** FERNANDO DE SANTIS Via Muro Maglie • ELETTROJOLLYCENTRO Via De Cesare, 13 • TEA Via Regina Elena, 101

CAMPANIA

ATRIPALDA FLIP FLOP Via Appia, 68 • **BENEVENTO** E.CO. INFORMATICA Via Pepicelli, 21/25 • **CASERTA** O.P.C. Via G.M. Bosco, 24 • MADDALONI M.P. COMPUTER Via Napoli, 30 • **NAPOLI** BABY TOYS Via Cisterna Dell'Olio, 5/bis • CASA MUSICALE RUGGIERO Int. Stazione Napoli • C.LE F.S. P.zza Garibaldi, 74 • CENTRO ELETTRONICO CAMPANO Via Epomeo, 121 • CI.AN Galleria Vanvitelli, 32 • CINE NAPOLI SNC Via Santa Lucia, 93/95 • DARVIN Calata San Marco, 26/25 • ELETTRONICA RO.DA.LO. Via Epomeo, 216/B • GIANCAR 2 P.zza Garibaldi, 37 • GRUPPO BUSCH Galleria Umberto, 55 • ODORINO L.go Lala, 22/A-B • R 2 SRL Via F. Cilea, 285 • TOP VIDEO - TOP COMPUTER VIA S. Anna Dei Lombardi, 12 • SPY Via San Giacomo Dei Capri, 36B/C • VIDEOFOTOMARKET Via S. Brigida, 19 • **S. ANASTASIA** (NA) SPADARO Via Romani, 93 • **CASORIA** ELECTRONIC DAY Via Delle Puglie, 17 • TUFANO S.S. Sannittica, 87 • **CASTELAMARE DI STABIA** SOF SUD V.le Europa, 59 • **FRATTAMAGGIORE** ELETTRONICA 2000 C.so Durante, 40 • **MUGNANO** GATEWAY Via Napoli, 68 • **PORTICI** NUOVA INFORMATICA SHOP Via Libertà, 185/191 • **POZZUOLI** BASIC COMPUTER C.so Garibaldi, 34 • **STRIANO** FALCO ELETTRONICA Via Sarno, 100 • **TORRE ANNUNZIATA** TECNOTRE Via P. Fusco, 1/F • **TORRE DEL GRECO** (NA) TECNO Via Vittorio Veneto, 28 • **SALERNO** COMPUTER MARKET C.so Vittorio Emanuele, 23 • **BATTIPAGLIA** (SA) KING COMPUTER Via Olevano, 56 • **EBOLI** (SA) DIMER POINT Via C. Rosselli, 20

CALABRIA

CATANZARO C. & G. COMPUTER Via F. Acri, 28 • PAONE SAVERIO & FIGLI Via F. Acri, 93/99 • **CROTONE** COMPUTER HOUSE Via Bologna (L.go Ospedale) • **VIBO VALENTIA** OTTICA FOTO NELLO RUELLO C.so Vittorio Emanuele,177 • **COSENZA** SIRANGELO COMPUTER Via Parisio, 25 • HI-FI ALFANO GIUSEPPE Via Baldacchini, 109 • **CASTROVILLARI** AMANTIA ELIGIO ANNICCHIARICO &C. SAS Via Roma, 21 • **CORIGLIANO** SCALO ALFA COMPUTER Via Nazionale, 341/A • **LAMEZIA TERME** ING. FUSTO SALVATORE C.so Nicotera, 99 • **REGGIO CALABRIA** CONTROL SYSTEM Via San Francesco Da Paola, 49D/E • SYSTEM HOUSE Via Fiume ang. Palestino, 1 • **LOCRI** (RC) COMPUTER SHOP V.le Matteotti, 50-52 • **TAURIANOVA** PICIEFFE SNC C.so F. Sofia Alessio

ESIGI SEMPRE LA GARANZIA DELLA COMMODORE ITALIANA

TRANSACTOR PER AMIGA N.3

SERVIZIO LETTORI **Compilare e spedire in busta chiusa a: GRUPPO EDITORIALE JACKSON Area Consumer - Via Rosellini, 12 - 20124 Milano**

A) Come giudichi questo numero di Transactor per Amiga?

- ☐ Ottimo
- ☐ Molto Buono
- ☐ Buono
- ☐ Discreto
- ☐ Sufficiente
- ☐ Insufficiente

B) Quale(i) articolo(i) o rubrica hai apprezzato di più?

Quale meno?

C) Cosa ti piacerebbe leggere nei prossimi numeri di Transactor per Amiga?

E) Quante persone leggono la tua copia di Transactor per Amiga?

F) Che computer possiedi?

Quale(i) computer intendi acquistare in futuro?

G) Leggi altre riviste Jackson?
☐ SI ☐ NO
Quali?

H) Leggi altre riviste dedicate al tuo computer?
☐ SI ☐ NO
Quali?

I) Oltre alle riviste dedicate al computer quali sono le tue letture preferite?

L) Quali sono i tuoi hobbies e maggiori interessi?

- ☐ Sport
- ☐ Fotografia
- ☐ Musica
- ☐ Automobile
- ☐ Videoregistrazione
- ☐ Moto
- ☐ Hi-Fi
- ☐ Viaggi

Altro

Nome

Cognome

Indirizzo

Età **Professione**

Città

Prov. **C.a.p.** **Tel.**

CEDOLE SERVIZIO LETTORI

ABBONAMENTO RIVISTE JACKSON

ATTENZIONE Questa cartolina riporta sul retro un modulo speciale con una serie di domande a cui preghiamo vivamente di rispondere con precisione.

ORDINI QUI SOTTO CIÒ CHE HA SCELTO

AVVERTENZA I rinnovi entreranno in vigore automaticamente a partire dal numero successivo alla data di scadenza dell'abbonamento precedente.
Per motivi tecnici, i nuovi abbonamenti saranno attivati dopo 6 settimane dalla data di ricevimento della cartolina di richiesta, o in data successiva nel caso di esplicita comunicazione da parte dell'abbonato.

N.B. Per abbonamenti all'estero le tariffe dovranno essere raddoppiate. Non è prevista la spedizione via aerea

☐ NUOVO ABBONAMENTO	☐ RINNOVO	
☐ EO NEWS SETTIMANALE	numeri 40 + 6 omaggio	L. 59.500
☐ ELETTRONICA OGGI	numeri 20	L. 80.500
☐ AUTOMAZIONE OGGI	numeri 20	L. 80.000
☐ MECCANICA OGGI	numeri 11	L. 36.000
☐ STRUMENTAZIONE E MISURE OGGI	numeri 11	L. 52.000
☐ INFORMATICA OGGI SETTIMANALE	numeri 40 + 6 omaggio	L. 61.000
☐ INFORMATICA OGGI MESE	numeri 11	L. 44.500
☐ BIT	numeri 11	L. 48.000
☐ PC MAGAZINE	numeri 11	L. 43.500
☐ PC FLOPPY	numeri 11	L. 106.000
☐ COMPUTERGRAFICA & DESKTOP PUBLISHING	numeri 11	L. 52.500
☐ TRASMISSIONE DATI E TELECOM.	numeri 11	L. 45.500
☐ NTE COMPUSCUOLA	numeri 10	L. 32.500
☐ WATT	numeri 20	L. 36.500
☐ MEDIA PRODUCTION	numeri 11	L. 62.000
☐ STRUMENTI MUSICALI	numeri 11	L. 44.000
☐ FARE ELETTRONICA	numeri 12	L. 58.000
☐ AMIGA MAGAZINE	numeri 11	L. 123.500
☐ AMIGA TRANSACTOR	numeri 6	L. 34.000
☐ SUPERCOMMODORE 64/128 disk	numeri 11	L. 105.500
☐ SUPERCOMMODORE 64/128 tape	numeri 11	L. 66.000
☐ OLIVETTI PRODEST USER	numeri 6	L. 24.000
☐ PC SOFTWARE	numeri 11	L. 88.000
☐ PC GAMES 5¼"	numeri 11	L. 124.000
☐ PC GAMES 3½"	numeri 11	L. 132.500
☐ 3½" SOFTWARE	numeri 11	L. 132.000

MODALITÀ DI PAGAMENTO

☐ Allego assegno n. _____ di L. _____

Banca _____

☐ Ho effettuato versamento di L. _____ sul c/c postale n. 11666203 intestato a Gruppo Editoriale Jackson - Milano e allego fotocopia della ricevuta.

☐ Ho effettuato versamento di L. _____ tramite vaglia postale o telegrafico e allego fotocopia della ricevuta.

☐ Vi autorizzo ad addebitare l'importo di L. _____ sulla carta di credito ☐ VISA ☐ AMERICAN EXPRESS ☐ DINERS CLUB ☐ CARTA SI

N. _____ Data di scadenza a _____

Data _____ Firma _____

COMMISSIONE LIBRI

ATTENZIONE Questa cartolina riporta sul retro un modulo speciale con una serie di domande a cui preghiamo vivamente di rispondere con precisione.

ORDINI QUI SOTTO CIÒ CHE HA SCELTO

Codice articolo	Titolo abbreviato	Prezzo L.
	Contributo fisso per spese di spedizione	L. 4.000
	TOTALE	

Ordine minimo £ 50.000 + £ 4.000 per contributo fisso spese di spedizione

☐ **Non sono abbonato** a riviste Jackson

☐ **Sono abbonato** alla/e seguente/i rivista/e Jackson _____

_____ e ho quindi diritto allo sconto del 20% sino al 28/2/89 e del 10% dopo tale data

MODALITÀ DI PAGAMENTO

☐ Contro assegno postale ☐ Assegno allegato n° _____

Banca _____

☐ Versamento su c/c postale 11666203 a Voi intestato del quale allego fotocopia della ricevuta

☐ Versamento con vaglia postale a Voi intestato del quale allego fotocopia della ricevuta

☐ Carta di Credito ☐ American Express ☐ Visa ☐ Diners Club

☐ Carta Sì Autorizzo l'organizzazione sopra indicata ad addebitare l'importo sul mio Conto n° _____ Scadenza _____

☐ Richiedo l'emissione della fattura (formula riservata alle aziende) e comunico il numero di Partita I.V.A. _____

Data _____ Firma _____

ARRETRATI RIVISTE

ATTENZIONE Questa cartolina riporta sul retro un modulo speciale con una serie di domande a cui preghiamo vivamente di rispondere con precisione.

ORDINI QUI SOTTO CIÒ CHE HA SCELTO

☐ INDUSTRIA OGGI	Numero _____	Anno _____	L. 10.000 cad.
☐ ELETTRONICA OGGI	Numero _____	Anno _____	L. 10.000 cad.
☐ AUTOMAZIONE OGGI	Numero _____	Anno _____	L. 10.000 cad.
☐ EO NEWS SETTIMANALE	Numero _____	Anno _____	L. 4.000 cad.
☐ WATT	Numero _____	Anno _____	L. 4.000 cad.
☐ TRASMISSIONE DATI E TELECOMUNICAZIONI	Numero _____	Anno _____	L. 10.000 cad.
☐ VIDEOTEL MAGAZINE	Numero _____	Anno _____	L. 8.000 cad.
☐ INFORMATICA OGGI	Numero _____	Anno _____	L. 10.000 cad.
☐ INFORMATICA OGGI SETTIMANALE	Numero _____	Anno _____	L. 4.000 cad.
☐ LAB NEWS	Numero _____	Anno _____	L. 10.000 cad.
☐ COMPUTER GRAFICA E APPLICAZIONI	Numero _____	Anno _____	L. 12.000 cad.
☐ PC MAGAZINE	Numero _____	Anno _____	L. 10.000 cad.
☐ PC MAGAZINE + PC FLOPPY	Numero _____	Anno _____	L. 24.000 cad.
☐ PC GAMES 5 1/4	Numero _____	Anno _____	L. 24.000 cad.
☐ PC GAMES 3½	Numero _____	Anno _____	L. 30.000 cad.
☐ PC 3½ SOFTWARE	Numero _____	Anno _____	L. 30.000 cad.
☐ BIT	Numero _____	Anno _____	L. 10.000 cad.
☐ SUPERCOMMODORE 64/128 (cassetta)	Numero _____	Anno _____	L. 15.000 cad.
☐ SUPERCOMMODORE 64/128 (disco)	Numero _____	Anno _____	L. 24.000 cad.
☐ NOI 128/64 (cassetta)	Numero _____	Anno _____	L. 18.000 cad.
☐ NOI 128/64 (disco)	Numero _____	Anno _____	L. 26.000 cad.
☐ COMMODORE PROFESSIONAL (cassetta)	Numero _____	Anno _____	L. 18.000 cad.
☐ COMMODORE PROFESSIONAL (disco)	Numero _____	Anno _____	L. 26.000 cad.
☐ AMIGA MAGAZINE (disco)	Numero _____	Anno _____	L. 28.000 cad.
☐ AMIGA TRANSACTOR	Numero _____	Anno _____	L. 14.000 cad.
☐ OLIVETTI PRODEST USER	Numero _____	Anno _____	L. 10.000 cad.
☐ LA RIVISTA DI ATARI	Numero _____	Anno _____	L. 10.000 cad.
☐ COMPUSCUOLA	Numero _____	Anno _____	L. 8.000 cad.
☐ FARE ELETTRONICA	Numero _____	Anno _____	L. 10.000 cad.
☐ LABORATORIO DI ELETTRONICA PROFESSIONALE	Numero _____	Anno _____	L. 3.000 cad.
☐ STRUMENTI MUSICALI	Numero _____	Anno _____	L. 10.000 cad.
☐ NAUTICAL QUARTERLY	Numero _____	Anno _____	L. 20.000 cad.
☐ AUTOMOBILE QUARTERLY	Numero _____	Anno _____	L. 20.000 cad.

MODALITÀ DI PAGAMENTO

NB: Non si effettuano spedizioni contro assegno il pagamento deve sempre essere allegato all'ordine

☐ Allego assegno n° _____ di L. _____

Banca _____

☐ Allego l'importo in contanti di L. _____

☐ Versamento su c/c postale 11666203 a Voi intestato del quale allego fotocopia della ricevuta

GRUPPO EDITORIALE JACKSON

COMPILARE E SPEDIRE IN BUSTA CHIUSA A: GRUPPO EDITORIALE JACKSON - VIA ROSELLINI 12 - 20124 MILANO

GRUPPO EDITORIALE JACKSON

SERVIZIO QUALIFICAZIONE LETTORI

ATTENZIONE Questa cartolina riporta un modulo speciale con una serie di domande a cui preghiamo vivamente di rispondere con precisione.

INDIRIZZO PRIVATO

COGNOME E NOME _____

VIA E NUMERO _____

CAP _____ CITTÀ _____ PROV. _____

TEL. (_____) _____ ANNO DI NASCITA 19____

TITOLO DI STUDIO: ☐ LAUREA ☐ MEDIA SUPERIORE ☐ MEDIA INFERIORE

INDIRIZZO LAVORO

DITTA O ENTE _____

VIA E NUMERO _____

CAP _____ CITTÀ _____ PROV. _____

TEL. (_____) _____ TELEX _____

ATTIVITÀ AZIENDA

- **A** ☐ Informatica
- **B** ☐ Automazione Industriale
- **C** ☐ Meccanica
- **D** ☐ Strumentazione elettronica
- **E** ☐ Telecomunicazioni e Telefonia
- **F** ☐ Elettronica
- **G** ☐ Chimica
- **H** ☐ Elettrotecnica e Impianti elettrici
- **I** ☐ Laboratori di analisi Chimica e medica
- **L** ☐ Altra industria manifatturiera
- **M** ☐ Agricoltura
- **N** ☐ Ingegneria/Edilizia/Architettura
- **O** ☐ Finanza/Banche/Assicurazioni
- **P** ☐ Editoria/Grafica/Pubblicità
- **Q** ☐ Pubblica amministrazione centrale/Locale
- **R** ☐ Consulenza legale/Commerciale
- **S** ☐ Commercio/Distribuzione
- **T** ☐ Istruzione (Scuola/Università)
- **U** ☐ Formazione/Ricerca
- **V** ☐ Broadcast/Audio e video professionale
- **Z** ☐ Strumenti musicali
- **X** ☐ Altro (specificare)

N. DI DIPENDENTI

- **A** ☐ da 1 a 49
- **B** ☐ da 50 a 249
- **C** ☐ da 250 a 999
- **D** ☐ da 1000 in su

FUNZIONI

- **AA** ☐ Acquisti
- **BB** ☐ Vendite
- **CC** ☐ Progettazione/Ricerca e sviluppo
- **DD** ☐ Marketing e Comunicazione
- **EE** ☐ Direzione Generale
- **FF** ☐ Produzione
- **GG** ☐ Amministrazione/ Personale/Finanza
- **HH** ☐ EDP
- **II** ☐ Professionista/Consulente
- **LL** ☐ Docente/Formatore
- **MM** ☐ Studente
- **NN** ☐ Altro (specificare)

INTERESSI PRINCIPALI

- **01** ☐ EDP
- **02** ☐ Personal Computer
- **03** ☐ Home Computer
- **04** ☐ Automazione Industriale e Meccanica
- **05** ☐ Strumentazione elettronica
- **06** ☐ Telecomunicazioni e telefonia
- **07** ☐ Elettronica professionale
- **08** ☐ Elettronica hobbystica
- **09** ☐ Elettrotecnica e impianti elettrici
- **10** ☐ Strumenti musicali
- **11** ☐ Marketing e management
- **12** ☐ Broadcast/audio e video professionale
- **13** ☐ Didattica
- **14** ☐ Altro (specificare)

CHE PERSONAL COMPUTER POSSIEDE

- **DOS** ☐ MS DOS e compatibili
- **MAC** ☐ Macintosh
- **AMG** ☐ Amiga
- **C64** ☐ Commodore 64
- **VAR** ☐ Altro home computer

COMPILARE E SPEDIRE IN BUSTA CHIUSA A: GRUPPO EDITORIALE JACKSON - VIA ROSELLINI 12 - 20124 MILANO

GRUPPO EDITORIALE JACKSON

SERVIZIO QUALIFICAZIONE LETTORI

ABBONAMENTO GRATUITO

A 6 NUMERI, A SCELTA TRA LE SEGUENTI RIVISTE SETTIMANALI

☐ EO News Sett. ☐ INFORMATICA Oggi Sett.

BARRARE LA CASELLA RELATIVA ALLA RIVISTA PRESCELTA

INDIRIZZO PRIVATO

COGNOME E NOME _____

VIA E NUMERO _____

CAP _____ CITTÀ _____ PROV. _____

TEL. (_____) _____ ANNO DI NASCITA 19____

TITOLO DI STUDIO: ☐ LAUREA ☐ MEDIA SUPERIORE ☐ MEDIA INFERIORE

INDIRIZZO LAVORO

DITTA O ENTE _____

VIA E NUMERO _____

CAP _____ CITTÀ _____ PROV. _____

TEL. (_____) _____ TELEX _____

ATTIVITÀ AZIENDA

- **A** ☐ Informatica
- **B** ☐ Automazione Industriale
- **C** ☐ Meccanica
- **D** ☐ Strumentazione elettronica
- **E** ☐ Telecomunicazioni e Telefonia
- **F** ☐ Elettronica
- **G** ☐ Chimica
- **H** ☐ Elettrotecnica e Impianti elettrici
- **I** ☐ Laboratori di analisi Chimica e medica
- **L** ☐ Altra industria manifatturiera
- **M** ☐ Agricoltura
- **N** ☐ Ingegneria/Edilizia/Architettura
- **O** ☐ Finanza/Banche/Assicurazioni
- **P** ☐ Editoria/Grafica/Pubblicità
- **Q** ☐ Pubblica amministrazione centrale/Locale
- **R** ☐ Consulenza legale/Commerciale
- **S** ☐ Commercio/Distribuzione
- **T** ☐ Istruzione (Scuola/Università)
- **U** ☐ Formazione/Ricerca
- **V** ☐ Broadcast/Audio e video professionale
- **Z** ☐ Strumenti musicali
- **X** ☐ Altro (specificare)

N. DI DIPENDENTI

- **A** ☐ da 1 a 49
- **B** ☐ da 50 a 249
- **C** ☐ da 250 a 999
- **D** ☐ da 1000 in su

FUNZIONI

- **AA** ☐ Acquisti
- **BB** ☐ Vendite
- **CC** ☐ Progettazione/Ricerca e sviluppo
- **DD** ☐ Marketing e Comunicazione
- **EE** ☐ Direzione Generale
- **FF** ☐ Produzione
- **GG** ☐ Amministrazione/ Personale/Finanza
- **HH** ☐ EDP
- **II** ☐ Professionista/Consulente
- **LL** ☐ Docente/Formatore
- **MM** ☐ Studente
- **NN** ☐ Altro (specificare)

INTERESSI PRINCIPALI

- **01** ☐ EDP
- **02** ☐ Personal Computer
- **03** ☐ Home Computer
- **04** ☐ Automazione Industriale e Meccanica
- **05** ☐ Strumentazione elettronica
- **06** ☐ Telecomunicazioni e telefonia
- **07** ☐ Elettronica professionale
- **08** ☐ Elettronica hobbystica
- **09** ☐ Elettrotecnica e impianti elettrici
- **10** ☐ Strumenti musicali
- **11** ☐ Marketing e management
- **12** ☐ Broadcast/audio e video professionale
- **13** ☐ Didattica
- **14** ☐ Altro (specificare)

CHE PERSONAL COMPUTER POSSIEDE

- **DOS** ☐ MS DOS e compatibili
- **MAC** ☐ Macintosh
- **AMG** ☐ Amiga
- **C64** ☐ Commodore 64
- **VAR** ☐ Altro home computer

COMPILARE E SPEDIRE IN BUSTA CHIUSA A: GRUPPO EDITORIALE JACKSON - VIA ROSELLINI 12 - 20124 MILANO

GRUPPO EDITORIALE JACKSON

SERVIZIO QUALIFICAZIONE LETTORI

ABBONAMENTO GRATUITO

A 6 NUMERI, A SCELTA TRA LE SEGUENTI RIVISTE SETTIMANALI

☐ EO News Sett. ☐ INFORMATICA Oggi Sett.

BARRARE LA CASELLA RELATIVA ALLA RIVISTA PRESCELTA

INDIRIZZO PRIVATO

COGNOME E NOME _____

VIA E NUMERO _____

CAP _____ CITTÀ _____ PROV. _____

TEL. (_____) _____ ANNO DI NASCITA 19____

TITOLO DI STUDIO: ☐ LAUREA ☐ MEDIA SUPERIORE ☐ MEDIA INFERIORE

INDIRIZZO LAVORO

DITTA O ENTE _____

VIA E NUMERO _____

CAP _____ CITTÀ _____ PROV. _____

TEL. (_____) _____ TELEX _____

ATTIVITÀ AZIENDA

- **A** ☐ Informatica
- **B** ☐ Automazione Industriale
- **C** ☐ Meccanica
- **D** ☐ Strumentazione elettronica
- **E** ☐ Telecomunicazioni e Telefonia
- **F** ☐ Elettronica
- **G** ☐ Chimica
- **H** ☐ Elettrotecnica e Impianti elettrici
- **I** ☐ Laboratori di analisi Chimica e medica
- **L** ☐ Altra industria manifatturiera
- **M** ☐ Agricoltura
- **N** ☐ Ingegneria/Edilizia/Architettura
- **O** ☐ Finanza/Banche/Assicurazioni
- **P** ☐ Editoria/Grafica/Pubblicità
- **Q** ☐ Pubblica amministrazione centrale/Locale
- **R** ☐ Consulenza legale/Commerciale
- **S** ☐ Commercio/Distribuzione
- **T** ☐ Istruzione (Scuola/Università)
- **U** ☐ Formazione/Ricerca
- **V** ☐ Broadcast/Audio e video professionale
- **Z** ☐ Strumenti musicali
- **X** ☐ Altro (specificare)

N. DI DIPENDENTI

- **A** ☐ da 1 a 49
- **B** ☐ da 50 a 249
- **C** ☐ da 250 a 999
- **D** ☐ da 1000 in su

FUNZIONI

- **AA** ☐ Acquisti
- **BB** ☐ Vendite
- **CC** ☐ Progettazione/Ricerca e sviluppo
- **DD** ☐ Marketing e Comunicazione
- **EE** ☐ Direzione Generale
- **FF** ☐ Produzione
- **GG** ☐ Amministrazione/ Personale/Finanza
- **HH** ☐ EDP
- **II** ☐ Professionista/Consulente
- **LL** ☐ Docente/Formatore
- **MM** ☐ Studente
- **NN** ☐ Altro (specificare)

INTERESSI PRINCIPALI

- **01** ☐ EDP
- **02** ☐ Personal Computer
- **03** ☐ Home Computer
- **04** ☐ Automazione Industriale e Meccanica
- **05** ☐ Strumentazione elettronica
- **06** ☐ Telecomunicazioni e telefonia
- **07** ☐ Elettronica professionale
- **08** ☐ Elettronica hobbystica
- **09** ☐ Elettrotecnica e impianti elettrici
- **10** ☐ Strumenti musicali
- **11** ☐ Marketing e management
- **12** ☐ Broadcast/audio e video professionale
- **13** ☐ Didattica
- **14** ☐ Altro (specificare)

CHE PERSONAL COMPUTER POSSIEDE

- **DOS** ☐ MS DOS e compatibili
- **MAC** ☐ Macintosh
- **AMG** ☐ Amiga
- **C64** ☐ Commodore 64
- **VAR** ☐ Altro home computer

SCOPRILO FINO ALL'ULTIMO BIT

Scopri tutto quello che può darti il tuo computer AMIGA.
Ogni mese, dal GRUPPO EDITORIALE JACKSON, tre riviste che ti danno di più: AMIGA TRANSACTOR per i programmatori più smaliziati, AMIGA MAGAZINE per chi vuole conoscere il proprio computer sempre di più e AMIGA MAGAZINE GAMES per i giocatori più accaniti.
AMIGA TRANSACTOR, AMIGA MAGAZINE e AMIGA MAGAZINE GAMES: un panorama completo ed esauriente dell'universo di AMIGA, tre riviste per usare il tuo computer fino all'ultimo bit.

GRUPPO EDITORIALE JACKSON
AREA CONSUMER

Scegli il meglio: scegli Jackson