LA RIVISTA PER I PROGRAMMATORI DI AMIGA — GENNAIO - FEBBRAIO 1989: NUMERO 2

L. 7.000 - S.FR. 10,50

Amiga Transactor

PER

EDIZIONE ITALIANA

Informazioni sui .Info • Breakpoint, parte seconda

L'ultima visita all'Arp Library • Lo standard ANSI

Uno sguardo alla struttura dei dischi di Amiga

Come internazionalizzare i vostri programmi • ARexx

Principi di Ray Tracing • Routine in AmigaBasic

per disegno 3D • Come cambiare il "Mouse Pointer"

in AmigaBasic

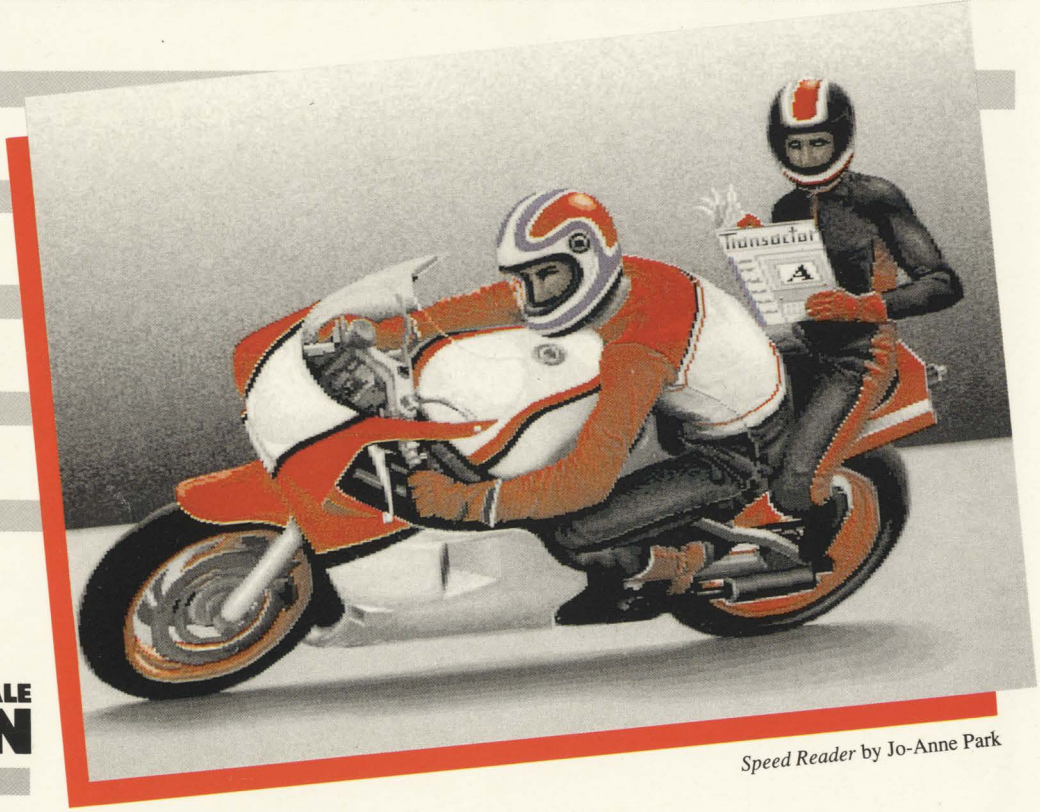
Il significato

dei colori •

Warrior cycles

Il Manx C3.6

e l'SDB



Speed Reader by Jo-Anne Park

GRUPPO EDITORIALE JACKSON
AREA CONSUMER

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

SIOA

Salone dell'Informatica della Telematica e della Organizzazione Aziendale

Fiera di Bologna
1-5 Aprile 1989

tecnobanca

Salone delle tecnologie e dei servizi per le attività bancarie, assicurative e finanziarie

futurabank

La prima simulazione attiva dello sportello bancario del futuro

Un'unità dimostrativa integrata

- i servizi all'utenza
- le tecnologie avanzate
- l'organizzazione
- il lay out

Organizzazione: Consorzio SIOA - Via Napoli, 20 - 40139 Bologna - Tel. 051/452936-466911 - Tlx. 583131 - Fax. 051/465234

Il Moro di Canale

Dio me ne scampi!
Non è certo questa la sede più adatta per parlare del celebre Moro del Canal Grande: l'Otello celebrato da Shakespeare.
Il mio (e nostro) Moro non è altri che una persona, realmente esistente, che vive in quel di Canale, un paese in provincia di Cuneo situato quasi a metà strada tra Asti e Alba.
Per alterne vicende della vita mi è capitato di vivere in quel paese per diverso tempo e proprio in tale periodo ho potuto osservare la maturazione, informatica s'intende, del nostro Moro. Ho ancora vivo il ricordo di una estate passata a decantare le virtù del nostro Amiga a un buon numero di persone, sia di Canale che dei dintorni. Il fatto poi che il Moro mi offrisse sempre un boccale di birra per iniziare a parlare, mi faceva tornare alla mente la scena di un film Western in cui gli avventori di un Saloon offrivano un bicchiere di Whisky al forestiero per poter ascoltare dalle sue parole cosa mai fosse successo nel lontano Ovest, facendomi sembrare, forse ai miei occhi soltanto, un novello oracolo della verità informatica.
Non so quando, convinsi il Nostro ad acquistare il "Favoloso Amiga", credo che la moglie maledica ancora quel giorno, rendendolo partecipe di quell'avventura che avevo vissuto fin dal lontano febbraio '86 con un Amiga NTSC. Cosa convinse il Moro a buttare il suo amatissimo C64 (con Speed DOS, digitalizzatori e ogni altra diavoleria) per passare al nuovo Amiga?
Prima di tutto il fascino delle cose nuove e belle: non c'è niente di meglio che scoprire ogni giorno qualcosa di diverso e che fino ad allora si riteneva impossibile. Poi perché il software (copiato, naturalmente) si trova molto facilmente. In realtà il primo programma che ha usato su Amiga lo ha comprato (era Logistix) pagando 180.000 lire e si è ritrovato in mano un programma più brutto di quelli che girano su IBM e per giunta non perfettamente funzionante. Dopo quello che gli avevo raccontato delle ottime capacità grafiche di Amiga, credo che la sua delusione sia stata veramente cocente. Poi ha comprato qualche libro, letto qualche rivista e ha disegnato qualche cosa con il DeLuxe Paint.
Cosa conosce ora di Amiga? Lo ha mai programmato? Come lo usa?
Probabilmente conosce bene il DOS, ha fatto qualche programmino in Basic, sa che esistono le librerie e i device, che Amiga è multitasking ma credo si sia fermato qui. Penso che abbia comprato un libro sul C ma che l'abbia anche presto dimenticato sullo scaffale.
Alla fine di novembre sono passato da casa sua per portargli un saluto veloce: manco a dirlo era davanti al suo amatissimo Amiga, ma non per usarlo, solo per copiare un dischetto per l'amico di turno.
Questa storia, lo ripeto è vera, è solo per denunciare la situazione in cui si trovano la maggior parte degli utenti di Amiga, e prima di loro quelli del C64, italiani: il computer che avete in mano è veramente eccezionale, se solo iniziaste a usarlo veramente. Certo non è facile imparare il C o l'Assembly, ma anche in Basic si riescono a fare cose soddisfacenti. La documentazione per l'Amiga è scarsa o illeggibile? E' vero, ma la situazione sta cambiando e credo che Amiga Transactor possa darvi una mano. Quello che manca è solo una piccola dose di coraggio. Sentirete che soddisfazione ad aver prodotto qualcosa di proprio, oppure ad aver scoperto qualche funzione strana, piuttosto che dimostrare all'amico di essere più bravo di lui a copiare i programmi. E se pensate che il vostro sforzo, sottoforma di programma, articolo o altro, possa essere condiviso dagli utenti italiani, mandatecelo; vedremo di pubblicarlo.
Moro, svegliati!

Marco Ottolini

per AMIGA
Transactor
La rivista dei programmatori di Amiga

Sommario

DIRETTORE RESPONSABILE
Giampietro Zanga

DIRETTORE EDITORIALE
Daniele Comboni

DIRETTORE TECNICO
Marco Ottolini

TRADUZIONI:
Leonardo Fei, Paolo Toccaceli, Giuseppe Bravo

COORDINAMENTO REDAZIONALE
Angelo Cattaneo

GRAFICA
Gianni De Tomasi

IMPAGINAZIONE ELETTRONICA
Piera Loddo

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, PUBBLICITA' E AMMINISTRAZIONE
via Rosellini, 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. 02:61222527-6187376

SEDE LEGALE
via Pietro Mascagni, 14 - 20122 MILANO

PREZZO DELLA RIVISTA: L. 7000
NUMERI ARRETRATI: L. 14000
ABBONAMENTO ANNUO (6 numeri): L. 25500
ESTERO: L. 51000

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)

LETTERE... ... E BIT

In questo numero la sezione dedicata alle lettere e ai bit è un po' scarna in quanto fino all'ultimo abbiamo sperato che qualche lettore italiano avesse mandato il suo contributo. Evidentemente la colpa è stata nostra in quanto nel primo numero non avevamo esplicitamente richiesto l'intervento dei lettori. Lo facciamo ora, sperando di essere sommersi da lettere di consigli e di critiche (non mandateci complimenti!). Saranno molto ben accetti anche interventi per la sezione Bit.

m.o.

Problemi di compatibilità per l'Amiga 2000

Volevo intervenire riguardo ai problemi di incompatibilità che possono insorgere usando un Amiga 2000: non tutti i programmi che funzionano regolarmente su un 1000 con Kickstart 1.2 si comportano nello stesso modo su un 2000. Fino ad ora ho riscontrato problemi con Artic Fox, Halley Project, Archon e Diablo. In particolare sono piuttosto deluso da Diablo, visto che l'ho comprato proprio l'altra sera; comunque ho già scritto alla Classic Image per informarli del problema. Sono sicuro che, oltre ai già citati, esistano diversi programmi dotati di auto-loader che presentino difficoltà quando usati su un Amiga 2000.
Fate sapere ai vostri lettori che cosa li aspetta!

Joyce Hancock, St.Louis, Missouri

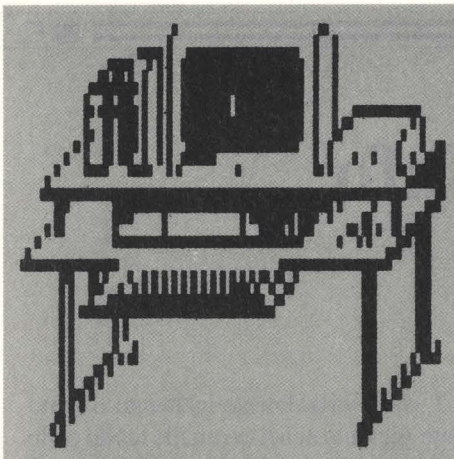
Sfortunatamente, questi e altri programmi non funzionano correttamente con i primi Amiga 2000. L'errore non risiede però nel prodotto Commodore visto che, per quanto a nostra conoscenza, il sistema operativo è identico per tutti gli Amiga attuali. I problemi di compatibilità come questi sono solitamente causati dall'uso diretto dell'hardware di Amiga, invece dell'uso dei meccanismi software creati all'uopo. Per esempio, nell'Amiga 2000 è stata modificata la temporizzazione della tastiera. Si tratta di una ragionevole modifica hardware che non dovrebbe in alcun modo interessare un'applicazione, sempre che rispette le regole fissate e non vada a usare direttamente l'hardware. La documentazione di Amiga è molto chiara riguardo a cosa un programma possa e non possa fare, quindi nessuno può incolpare la Commodore se ha operato una piccola modifica come questa. Non esiste nessuna garanzia che non ci saranno altri cambiamenti di questo tipo nei futuri Amiga: quindi se qualche programma non funzionerà a causa di ciò sarà perché è stato scritto male.
Queste informazioni non sono ovviamente di nessun aiuto per gli utenti, poiché dal loro punto di vista esiste solo il fatto che una parte della loro collezione di software non funziona più solo a causa dell'upgrade a un Amiga 2000. L'unica soluzione è che il produttore di software fornisca gratuitamente le nuove versioni dei programmi, visto che in ogni caso tali versioni devono essere sviluppate per soddisfare le richieste dei nuovi utenti di Amiga 2000. Speriamo che almeno tali situazioni insegnino ai programmatori di software commerciale qualcosa riguardo alla compatibilità, in modo che non ripetano gli stessi errori con eventuali Amiga del futuro.

Copia con un singolo drive
di Graham Reed

Avete mai desiderato formattare o copiare un disco avendo però già caricato il Workbench di un programma applicativo? Probabilmente avrete ricevuto in risposta dal Workbench un fastidioso "Can't open SYS:System/Format" (oppure DiskCopy). Il fatto che DiskCopy e Format vogliono che si usino le icone e i menu, non significa che in questi casi si debba ricorrere a scrivere qualcosa con il CLI, visto anche che spesso i programmi applicativi non ne prevedono l'uso.
Quindi, per risolvere l'arcano con un sistema dotato di un solo disco, bisogna porre un disco Workbench normale nel drive e aprire il cassetto System. Poi si deve sostituire il Workbench con il disco che si intende copiare o formattare e bisogna selezionarne l'icona relativa. Usando il metodo di selezione esteso, premete il tasto SHIFT ed effettuate un doppio click sull'icona Format o DiskCopy, a seconda di ciò che volete fare. Ora non resta che seguire le indicazioni dei Requester e introdurre il dischetto da formattare oppure quello da copiare e il duplicato.
Per i fortunati possessori di due drive il procedimento da seguire per formattare un dischetto è essenzialmente quello descritto, eccezion fatta per lo scambio tra il dischetto del Workbench e quello vergine. Per copiare un disco si deve invece aprire il cassetto System come prima per poi inserire il dischetto sorgente e destinazione nei due drive a disposizione. Ora, la cosa importante è l'ordine con il quale si selezionano le icone dei due dischetti poiché in questo modo si decide qual'è il disco sorgente e quello destinazione: proteggete quindi dalla scrittura il sorgente per maggior sicurezza. Selezionate l'icona del disco sorgente poi, tenendo premuto il tasto SHIFT selezionate anche l'icona destinazione. Sempre premendo lo SHIFT, effettuate un doppio click sull'icona DiskCopy. Seguite le indicazioni dei Requester come al solito e... Ecco fatto!

Seconda conferenza europea degli sviluppatori

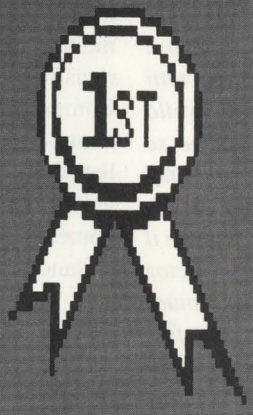
Dal 16 al 18 gennaio scorso si è svolta a Francoforte, in Germania Federale, la seconda conferenza europea degli sviluppatori di Amiga organizzata direttamente dalla Commodore per fornire un supporto diretto a tutti i programmatori professionisti impegnati con l'Amiga. Gli argomenti discussi sono stati tanti e tutti molto interessanti, ma quelli che hanno destato maggiore attenzione sono stati sicuramente quelli riguardanti i nuovi prodotti Commodore, il funzionamento dell'autoboot del Kickstart 1.3, nonché il trasporto di Unix su Amiga e l'interfacciamento con i Transputer.
Molto pittoresca è stata l'ultima sezione della conferenza dedicata alle proposte di modifiche per le versioni del sistema operativo oltre la 1.4 (virtualmente già pronta). Da segnalare la presenza tra gli speaker della conferenza dei nostri collaboratori Rob Peck e John Toebes. Per problemi di tempo, la conferenza è terminata il giorno prima dell'andata in stampa, riferiremo dei contenuti nel prossimo numero di Transactor per Amiga.



AMIGA™

COMMODORE

COMPUTER CENTER
030/223230



AMIGALINE BBS 24 h
030/2420452

Il Centro TUTTO COMMODORE

più qualificato e completo!

..impossibile chiedere di più..

..assurdo accontentarsi di meno!!

COMPUTER CENTER POINT

Via Cipro int. 62 - 25124 BRESCIA

Informazioni sui .Info

di Betty Clay

Visitiamo i misteriosi file icona

Betty Clay è una insegnante di matematica in pensione e le piace studiare i computer della Commodore. Può essere raggiunta via Compuserve (74145,657), QuantumLink (bjk), o via posta al 1322 South Oak Street, Arlington, Texas, 76010.

***Nota del redattore**: questo è un esame approfondito della struttura dei file ".info" che definiscono le icone. Perché preoccuparsi di cosa c'è in questi file? Normalmente si usano le routine di sistema per manipolare le Icone (GetDiskObject e PutDiskObject) quindi le specifiche di come siano composti questi file non sono estremamente importanti. Questa è la teoria, ma tutti sappiamo che la conoscenza delle cose al più profondo livello possibile facilita sempre la soluzione dei problemi, anche se questo vuole dire fare qualche veloce modifica direttamente nel codice binario anziché scrivere un programma apposito. In generale è più sicuro usare le routine di sistema per eseguire il nostro lavoro, ma è sempre stata una tradizione di Transactor rivelare in dettaglio come funzionano le cose, giusto per rimuovere qualsiasi velo di mistero che potrebbe potenzialmente frenarci nei nostri sforzi di programmazione. Per mantenerci fedeli alla tradizione vi presentiamo quindi la versione Transactorizzata dell' esame di Betty sui file Icona.*

La prima cosa che si impara su Amiga, sicuramente la più facile, consiste nel puntare il mouse su un'icona e cliccare. Sicuramente non c'è niente di più fondamentale nell'interfaccia utente standard di questa macchina. Ma avete provato a decifrare i file che contengono queste icone? La documentazione relativa è sparpagliata in vari punti del *ROM Kernel Manual* e l'ho trovata decisamente difficile da seguire. L'aiuto migliore ci viene dall'utilizzo combinato dei file "Intuition/Intuition.i" (o ".h") e "Workbench/Workbench.i" (o ".h"). Trovo che le descrizioni in linguaggio Assembly siano state molto più comprensibili di quelle in linguaggio C. Anche la determinazione nel raggiungere lo scopo e una buona immaginazione sono state di grande aiuto.

Andiamo a fare un giro nel disco del Workbench 1.2 per dare un'occhiata?

Il file ".info"

La prima cosa che si scopre sui file .info è che non sono tutti simili. Ogni tipo di file ha il suo formato. Il nostro viaggio inizia con il file che si chiama esclusivamente ".info". Questo ha un formato completamente diverso dagli altri file il cui nome termina con .info. Sembra che il suo compito principale consista nel contenere una lista di nomi degli altri file .info che si trovano nella stessa directory. Ecco una stampa in forma esadecimale del file ".info" del disco Workbench 1.2, che è virtualmente inalterato rispetto all'originale: ho solamente aggiunto nel disco un file molto corto creato con Notepad per avere un .info di tipo project da esaminare.

```
0000: F34C0012 00000E32 00000568 0000020C
```

Il formato del file ".info" non è documentato da nessuna parte, così è difficile fare qualcosa di più che indovinare il significato di questi primi byte. Queste informazioni non sarebbero comunque molto utili perché il formato sarà soggetto a cambiamenti nelle revisioni future del sistema operativo (ragione per la quale la Commodore-Amiga non l'ha documentato da principio). Il contenuto della parte rimanente del file, comunque, è più ovvio: una lista delle altre icone contenute in questa directory, tutte elencate nel leggibilissimo codice ASCII, con un linefeed ($0A) dopo ogni nome. Il RKM (ROM Kernel Manual) dice che qui è "dove abitano i nostri figli".

```
0010: 45787061 6E73696F 6E0A436C 6F636B0A
0020: 50726566 6572656E 6365730A 434C490A
0030: 44656D6F 730A5574 696C6974 6965730A
0040: 53797374 656D0A45 6D707479 0A547261
0050: 73686361 6E0A
```

Questo è l'equivalente ASCII dei byte precedenti (con un punto al posto del linefeed che separa ogni stringa):

```
Expansion.Clock.Preferences.CLI.Demos.
Utilities.System.Empty.Trashcan.
```

Il file .info ha una struttura estremamente semplice. Non ha bisogno di alcuna grafica perché non visualizza niente sullo schermo; non ha flag ed è composto quasi interamente di testo.

I file "xxxx.info", quelli che rappresentano Cassetti (Drawers), Applicazioni (Tools), Dischi (Disks), Progetti (Projects), e Pattumiere (Trashcans), hanno qualche struttura in comune, ma ci sono differenze a seconda dei tipi. Per ognuno di essi si comincia con una struttura 'DiskObject'. Nelle Icone di tipo Drawer, Disk e Trashcan segue, subito dopo, una struttura 'DrawerData' che definisce la finestra che viene aperta quando l'icona è selezionata con il doppio click. Segue una struttura 'Image' che definisce l'immagine dell'Icona, seguita a sua volta dai vari bitplane di dati che rappresentano l'immagine grafica. Nelle icone che hanno due immagini, una per il loro stato non selezionato e l'altra per il loro stato selezionato, segue una seconda struttura 'Image' con i relativi bitplane di dati. Infine, qualora vengano utilizzate le loro funzioni, seguono le stringhe 'DefaultTool' e 'ToolTypes'.

Questa è la composizione generica di un file icona standard nella release attuale del Workbench.

Una passeggiata attraverso Disk.info

Guardiamo da vicino il file "Disk.info". Questo è il file che serve a visualizzare l'icona del disco sullo schermo del Workbench. Quando selezioniamo l'icona del disco con il doppio click, verrà aperta una finestra nella quale saranno posizionate le varie icone (generalmente Drawers) presenti nella directory principale (root directory). Questo è il codice assembly che descrive le informazioni contenute nella struttura DiskObject:

```
STRUCTURE DiskObject,0
  UWORD     do_Magic        ;sempre $e310
  UWORD     do_Version      ;1 per questa verione del WB
   STRUCT   do_Gadget,gg_   SIZEOF ;
                            copia della struttura gadget
  UWORD     do_Type         EQU   1
    ;WBDISK                 EQU   2
    ;WBDRAWER               EQU   3
    ;WBTOOL                 EQU   4
    ;WBPROJECT              EQU   5
    ;WBGARBAGE              EQU   5
  APTR      do_DefaultTool  ;Tool (programma)
                            per questo file
  APTR      do_ToolTypes     ;argomenti del programma
  LONG      do_CurrentX      ;posizione X dell'Icona
  LONG      do_CurrentY      ;posizione Y dell'Icona
  APTR      do_DrawerData    ;descrive la finestra
;gli ultimi due sono usati solo per i Tool:
  APTR      do_ToolWindow    ;per le I/O del programma
  LONG      do_StackSize     ;default = 4K, può aumentare
```

Notiamo che la struttura do_Gadget è contenuta nella struttura DiskObject. Le Icone sono in realtà dei gadget di Intuition che vengono manipolati dal Workbench. Questa è la struttura Gadget:

```
 STRUCTURE Gadget,0
   APTR   gg_NextGadget   ;collegamento nella lista
   WORD   gg_LeftEdge     ;distanza dal bordo sinistro
   WORD   gg_TopEdge      ;distanza dal bordo in alto
   WORD   gg_Width        ;larghezza in pixel dell'icona
   WORD   gg_Height       ;altezza in linee dell'icona
   WORD   gg_Flags        ;highlight o immagine alternata
   WORD   gg_Activation   ;cosa attiva il gadget?
   WORD   gg_GadgetType   ;le icone sono BOOLEAN
   APTR   gg_GadgetRender ;ptr ai dati per l'immagine
   APTR   gg_SelectRender ;ptr all'immagine alternata
   APTR   gg_GadgetText   ;ptr al testo (non utilizzato)
   LONG   gg_MutualExclude
   APTR   gg_SpecialInfo
   WORD   gg_GadgetID
   APTR   gg_UserData
```

GadgetRender dovrebbe puntare a una struttura Image che definisce l'aspetto del Gadget, ma in questo file il puntatore sembra non avere significato, dal momento che la struttura Image è in una posizione conosciuta. Il campo SelectRender è invece significativo, in quanto se è diverso da zero significa che c'è una seconda struttura Image con i relativi bitplane di dati che segue la prima; ciò significa che l'Icona avrà un'immagine diversa quando verrà selezionata, anziché essere semplicemente evidenziata per mezzo di un cambiamento di colori. A parte questo, non c'è da guadagnarci nel modificare la struttura Gadget; non possiamo per esempio aggiungere del GadgetText e nemmeno modificare la posizione del Gadget (che viene determinata da do_CurrentX e do_CurrentY). Gli unici campi che sembrano avere un qualche effetto sull'icona sono Width, Height, Flags e SelectRender.

Probabilmente avrete notato che la struttura DiskObject contiene un puntatore alla struttura DrawerData. Questo puntatore viene usato internamente, ma non sembra essere correlato alla posizione della struttura DrawerData nel file. Comunque sia non ha importanza, dal momento che la struttura DrawerData (presente solo per le icone di tipo Disk, Drawer e Trashcan) segue sempre immediatamente dopo la struttura DiskObject. La struttura DrawerData stabilisce le caratteristiche della finestra che verrà aperta quando attiveremo l'icona del disco:

```
STRUCTURE DrawerData,0
  STRUCT    dd_NewWindow,DD_SIZEOF;
  LONG      dd_CurrentX;
  LONG      dd_CurrentY;
```

La prima cosa che troviamo nella struttura DrawerData è una completa struttura NewWindow. Chiunque abbia scritto un programma per aprire una finestra su Amiga dovrebbe avere familiarità con questa struttura. Come nel caso della struttura Gadget, il cambiare alcuni dati contenuti nella struttura NewWindow produce degli effetti limitati. Forse il più interessante si ottiene cambiando le variabili DetailPen e BlockPen in modo da fare apparire la finestra con colori diversi dal solito (per esempio in arancione e nero, anziché in blu e bianco).

Di seguito alla struttura DrawerData si trova la struttura Image che contiene il bitmap dell'icona. Vediamone il contenuto:

```
STRUCTURE Image,0
  WORD ig_LeftEdge      ;offset X dal bordo sinistro
  WORD ig_TopEdge       ;offset Y dalla cima
  WORD ig_Width         ;larghezza
  WORD ig_Height        ;altezza
  WORD ig_Depth         ;numero di bitplane usati
  APTR ig_ImageData     ;puntatore all'immagine
  BYTE ig_PlanePick     ;quali bitplane utilizzare
  BYTE ig_PlaneOnOff    ;riempimento di quelli non usati
  APTR ig_NextImage     ;prossima immagine
```

Cambiando la struttura Image si modificano le caratteristiche, le dimensioni e la posizione dell'immagine dell'icona, ma non del suo gadget.
Mettendo un valore diverso da zero nelle variabili LeftEdge e TopEdge, per esempio, si può spostare l'immagine dell'icona rispetto al rettangolo nel quale l'utente deve cliccare, creando una situazione confusa, ragione per cui questi due campi sono generalmente lasciati a zero.

Esaminiamo adesso una stampa esadecimale del file Disk.info, cercando di ritrovare le strutture appena viste in Assembly.

```
0000: E3100001 ........ ........ ........
```

$E310 - do_Magic, un numero che identifica questo file come file .info.

$0001 - do_Version. Questa è la versione 1 del Workbench. Se volete lavorare direttamente su questi file, assicuratevi per prima cosa che la versione corrisponda al numero 1, in quanto il formato potrebbe cambiare in una qualsiasi revisione futura, creando grandi problemi sia a noi sia al nostro programma.

A cominciare con la seconda longword, troviamo (come illustrato precedentemente) una struttura standard di tipo Gadget, nella quale alcuni campi non vengono utilizzati e risultano pertanto inizializzati a zero.

```
0000: ........ 00000000 021E0005 00200010
```

$00000000 gg_NextGadget - utilizzato dal sistema, punta al prossimo gadget
$021E gg_LeftEdge - offset del gadget dal bordo sinistro dello schermo
$0005 gg_TopEdge - offset del gadget dalla cima dello schermo
$0020 gg_Width - larghezza del gadget
$0010 gg_Height - altezza del gadget

```
0010: 00050003 00010000 C2180000 00000000
```

$0005 gg_Flags, GADGIMAGE ($0004) il gadget ha un'immagine e non un bordo, GADGBACKFILL ($0001 flag esclusivamente per i gadget delle icone) riempie la zona attorno al gadget con il colore di sfondo.

$0003 gg_Activation, GADGIMMEDIATE ($0002) genera un messaggio quando l'icona viene selezionata, REL VERIFY ($0001) genera un messaggio quando il pulsante del mouse viene rilasciato.

$0001 gg_GagdgetType. Le icone sono sempre gadget di tipo BOOLEAN (il tipo sul quale si clicca con il mouse).

$0000 C218 gg_GadgetRender, punta al bordo o all'immagine da visualizzare. Le icone utilizzano sempre un'immagine.

$0000 0000 gg_SelectRender, se è diversa da zero c'è un'altra immagine da visualizzare selezionando l'icona.

La parte rimanente della struttura Gadget non viene utilizzata per i file .info, quindi i campi seguenti sono tutti a zero:

$0000 gg_GadgetText. Questa è una longword che continua qui sotto.

```
0020: 00000000 00000000 00000000 00000000
```

$0000 parte rimanente della longword GadgetText precedente.
$0000 0000 gg_MutualExclude
$0000 0000 gg_SpecialInfo
$0000 gg_GadgetID
$0000 0000 gg_UserData

Avendo terminato la parte do_Gadget, ritorniamo al resto della struttura DiskObject:

```
0030: 013C0001 AB200000 00000000 021E0000
```

$013C do_Type, questa è un'icona di tipo WBDISK. Il numero $01 è la parte importante in quanto indica il tipo dell'icona (vedere le definizioni incontrate prima con la struttura DiskObject). L'altro byte di questa word viene utilizzato solo per allineare i dati con un indirizzo pari, quindi contiene un valore a caso e può essere ignorato.

$0001 AB20 do_DefaultTool. Puntatore al nome del file che rappresenta il default tool di questa icona. Esempio: per un file creato con il NotePad, il default tool sarà il NotePad. Per il file Disk.info il default tool è sempre il comando DiskCopy.

$0000 0000 do_ToolTypes - non utilizzata
$0000 021E do_CurrentX - posizione X dell'icona
$0000 do_CurrentY - posizione Y dell'icona (usa anche i prossimi 4 byte)

```
0040: 00050001 B8100000 00000000 00000032
```

$0000 0005 do_CurrentY - parte rimanente della posizione Y dell'icona
$0001 B810 do_DrawerData - puntatore alla struttura DrawerData

Poiché l'icona che stiamo esaminando non è di tipo TOOL, non c'è bisogno delle variabili ToolWindow e StackSize che sono quindi messe a zero.

$0000 0000 do_ToolWindow
$0000 0000 do_StackSize

Qui termina la struttura DiskObject. Segue la struttura DrawerData (che non sarebbe presente se questa icona fosse di tipo Tool o Project). La prima cosa che incontriamo adesso è una struttura NewWindow:

$0032 nw_LeftEdge - offset dal bordo sinistro dello screen

```
0050: 00320190 0064FFFF 00000000 0200027F
```

$0032 nw_TopEdge - offset dalla cima dello screen
$0190 nw_Width - larghezza iniziale
$0064 nw_Height - altezza iniziale
$FF nw_DetailPen - $FF (-1) segnala alle routine del sistema operativo di utilizzare il valore di default per questa variabile, che rappresenta il colore nel quale verranno disegnati il nome della finestra e i dettagli del disegno.

$FF nw_BlockPen - stesso discorso per questa variabile che rappresenta il colore utilizzato per il grosso del disegno.
$0000 0000 nw_IDCMPFlags - non utilizzato in questo file.
$0200 027F nw_Flags - Flag per le caratteristiche della finestra. Nel nostro caso sono stati utilizzati i seguenti flag: WINDOWSIZING, WINDOWDRAG, WINDOWDEPTH, WINDOWCLOSE, SIZEBRIGHT, SIZEBOTTOM, SIMPLE_REFRESH, NOCAREREFRESH e WBENCHWINDOW.

```
0060: 00033914 00000000 0000C240 00000000
```

$0003 3914 nw_FirstGadget - Il Workbench collegherà i propri gadget
$0000 0000 nw_CheckMark - non utilizzata
$0000 C240 nw_Title - Il Workbench visualizzerà il nome della finestra
$0000 0000 nw_Screen - non utilizzata

```
0070: 00000000 005A0028 FFFFFFFF 00010000
```

$0000 0000 nw_BitMap - non utilizzata
$005A nw_MinWidth - larghezza minima della finestra
$0028 nw_MinHeight - altezza minima della finestra
$FFFF nw_MaxWidth - larghezza massima della finestra
$FFFF nw_MaxHeight - altezza massima della finestra
$0001 nw_Type - tipo WBENCHSCREEN

Qui termina la struttura NewWindow. Segue la parte rimanente di DrawerData:

```
0080: 0000FFFF FFFE0000 00000020 00100002
```

$0000 0000 dd_CurrentX - la distanza sull'asse delle X delle icone in questa finestra (la prima word di questa longword viene dalla fine della linea 0070). Questa distanza viene modificata utilizzando la barra di scroll contenuta nel bordo della finestra, che ci permette di vedere parti differenti di un grosso insieme di icone.

$FFFF FFFE dd_CurrentY - la distanza sull'asse delle Y delle icone in questa finestra.

A questo punto termina la struttura DrawerData e comincia la struttura Image, che dice ad Amiga come disegnare l'icona standard per i dischi.

$0000 ig_LeftEdge - l'immagine è allineata con il bordo sinistro del gadget (do_CurrentX)
$0000 ig_TopEdge - ed è anche allineata con la cima di questo (do_CurrentY)
$0020 ig_Width - l'icona è larga 32 pixel,
$0010 ig_Height - alta 16 pixel,
$0002 ig_Depth - e impiega 2 bitplane (4 colori)

```
0090: 0001B9D0 03000000 00000000 ........
```

$0001 B9D0 ig_ImageData - punta ai dati per l'immagine (non ha significato nel file)
$03 ig_PlanePick - seleziona i bitplane 0 e 1
$00 ig_PlaneOnOff - non ci sono bitplanes inutilizzati
$0000 0000 ig_NextImage - c'è una sola immagine in questa icona

Adesso troviamo i dati per i due bitplane che descrivono l'immagine dell'icona. Ecco come appaiono quando i bit vengono rappresentati con dei punti al posto degli uno e con '#' al posto degli zero:

```
................................
..############....######........
..############....########......
..############....##########....
..############....############..
..##############################..
..##############################..
..##############################..
..##############################..
..##############################..
..##############################..
..##############################..
..##############################..
..##############################..
..##############################..
..##############################..
................................
```

Questa è la rappresentazione del bitplane numero zero, che costituisce il bit di ordine basso per la selezione del colore di ogni pixel dell'immagine. Adesso segue il bitplane numero 1 (cominciando a metà della terza longword della linea $00D0):

```
00D0: ........ ........ ....FFFF FFC0C0FF
```

```
############################......
##......##############..####....
##......##############....####..
##......##############......####
##......##############........###
##......##############..........##
##..............................##
##..............................##
##..............................##
##..............................##
##..............................##
##..............................##
##..............................##
##..............................##
################################
```

Notiamo che i pixel spenti nel piano 0 sono generalmente accesi nel piano 1. Se osserviamo l'icona del disco su uno Workbench con i colori standard, i bit accesi nel piano 1 saranno del colore che vediamo per i bordi (nero). Su un Workbench standard il colore zero è il blu, l'uno è bianco, il due nero e il tre arancio. In altre parole, il colore che vedremo per ogni pixel è determinato dai bit corrispondenti in ogni bitplane, seguendo questa regola:

Bitplane 0	Bitplane 1	Risultato colore
0	0	0 (blu)
0	1	1 (bianco)
1	0	2 (nero)
1	1	3 (arancio)

Dopo le mappe a bit il file termina con la stringa che identifica i tool di default e i "ToolTypes" (se ve ne sono):

```
0110: ........ ........ ........ ....5359
0120: 533A5379 7374656D 2F446973 6B436F70
0130: 7900
```

Tradotto in ASCII diventa una stringa terminante con zero:

```
..............SYS:System/DiskCopy
```

Questo è il programma che verrà lanciato quando l'icona del disco viene posizionato sopra l'icona di un altro disco per effettuare la copia.

Variazioni riscontrabili in altre icone

Il file "Empty.info", un'icona di tipo Drawer, è abbastanza simile al file "Disk.info", pur essendoci qualche differenza. Notiamo che la prima parte del file differisce solo per quanto riguarda le dimensioni e la posizione dell'icona. Inoltre la variabile Flag qui contiene $0004, indicando che per quest'icona non verranno cancellate le immagini preesistenti prima di essere disegnata al suo posto ("backfill"), poiché l'intera zona occupata dal gadget verrà cambiata di colore (highlight) quando verrà selezionata.

```
0000: E3100001 00000000 00820025 0040000D
0010: 00040003 00010001 17000000 00000000
0020: 00000000 00000000 00000000 00000000
```

Nel file "Disk.info" avevamo il numero che identifica un'icona di tipo Disk all'inizio della locazione $0030. In questo caso troviamo invece il numero che identifica il tipo Drawer (se osserviamo la definizione della struttura DiskObject, noteremo che il tipo WBDRAWER è uguale a due).

Segue la struttura DrawerData, cominciando con la struttura NewWindow. Qui le uniche differenze con il file Disk.info sono le dimensioni e la posizione della finestra.

```
0030: 02F40000 00000000 00000000 00700000
0040: 00180001 F0D80000 00000000 0000011C
0050: 005C00E6 004AFFFF 00000000 0240027F
0060: 0001F14C 00000000 0000AC90 00000000
0070: 00000000 005A0028 FFFFFFFF 00010000
```

Qui comincia la struttura Image. I dati descrivono naturalmente l'immagine di un cassetto, anziché quella di un disco. Possiamo vedere le dimensioni e la posizione dell'icona qui descritta. Notiamo che nell'ultima longword alla linea $0080 l'altezza dell'icona è inferiore di un pixel rispetto a quella specificata per il gadget. Questo succede perché il progettista ha scelto di lasciare una linea vuota sullo schermo, fra l'icona e il suo nome. Possiamo effettuare una scelta analoga in IconED quando specifichiamo il "bottom border".

```
0080: 00000000 00000000 00000040 000C0002
0090: 0001A458 03000000 00000000 00000000
```

Ed ecco i bitmap per ogni piano utilizzato nel disegno:

```
00A0: 00003FFF FFFFFFFF FFFC3C00 00000000
00B0: 003C3CFF FFFFFFFF FF3C3CFF FFFFFFFF
00C0: FF3C3CFF FF3FFCFF FF3C3CFF FF0000FF
00D0: FF3C3CFF FFFFFFFF FF3C3CFF FFFFFFFF
00E0: FF3C3C00 00000000 003C3FFF FFFFFFFF
00F0: FFFC0000 00000000 0000FFFF FFFFFFFF
0100: FFFFC000 00000000 0003C3FF FFFFFFFF
0110: FFC3C300 00000000 00C3C300 00000000
0120: 00C3C300 00C00300 00C3C300 00FFFF00
0130: 00C3C300 00000000 00C3C300 00000000
0140: 00C3C3FF FFFFFFFF FFC3C000 00000000
0150: 0003FFFF FFFFFFFF FFFF
```

Dal momento che non c'è un tool di default né un ToolType, il file termina alla fine del secondo bitmap.

Il file **"Notepad.info"** è un'icona di tipo Tool. Il file comincia con il solito "numero magico" che contraddistingue le icone, quindi segue la solita descrizione sulla posizione e sulle dimensioni.

```
0000: E3100001 00000000 0023000B 00210012
0010: 00050003 00010000 FA680000 00000000
0020: 00000000 00000000 00000000 00000000
0030: 03840000 00000000 00000000 001F0000
0040: 00000000 00000000 00000000 00000000
```

Qui il numero di identificazione del tipo di icona alla linea $0030 è uguale a $03 (WBTOOL). Notiamo che la variabile StackSize (dimensioni dello stack) contiene zero. Questo indica che verranno utilizzate le dimensioni di default fornite dal Workbench. Da notare anche che il puntatore DrawerData è a zero. Non esiste una struttura DrawerData per il tipo Tool, per cui a questo punto troviamo subito la struttura Image, seguita dai dati per i bitplane:

```
0050: 00000021 00110002 00027E38 03000000
0060: 00000F07 83C00000 180C0600 00002DF6
0070: FB400000 3FFFFFC8 00003FFC 27CA0000
0080: 3FFB87CA 00003FFE 27CA0000 3FF8E7CA
```

```
0090: 00003FE0 07CA0000 338FE7CA 0000263F
00A0: E3CA0000 30FFF0CA 00003FFF FFCA0000
00B0: 00000002 00000AAA AAA80000 00000000
00C0: 00000000 00000000 0F0783C0 0000FFFF
00D0: FFF00000 DE0F07BC 0000C000 00370000
00E0: C003D835 8000C004 78358000 C001D835
00F0: 8000C007 18358000 C01FF835 8000CC70
0100: 18358000 D9C01C35 8000CF00 0F358000
0110: C0000035 8000FFFF FFFD8000 35555557
0120: 80000FFF FFFE0000 00000000 0000
```

Questa è un'icona piuttosto elaborata, quindi i bitmap hanno un'aspetto differente da quelli che abbiamo visto prima. La larghezza dell'immagine è di 33 pixel, mentre l'altezza è di 17 linee. I dati per i bitplane cominciano alla linea $0060.

Non ci sono ToolTypes, quindi il file finisce qui.

Il file **"Project.info"**. Per rendere questo articolo il più completo possibile, ho creato un piccolo file chiamato "Testfile" utilizzando il Notepad. I dati che seguono rappresentano l'icona di tipo Project generata da Notepad per il file "Testfile". Questa è ovviamente l'icona standard per tutti i file creati da Notepad.

La parte iniziale di questo file differisce da tutti gli altri solo per le dimensioni e per la posizione dell'icona. Il resto rimane uguale. Questo gadget è posizionato a 97 pixel dal bordo sinistro, a 12 linee dalla cima. Le sue dimensioni sono 40 pixel di largezza e 15 linee di altezza.

```
0000: E3100001 00000000 0061000C 00280015
0010: 00040003 00010027 3FD20000 00000000
0020: 00000000 00000000 00000000 00000000
```

Alla linea $0030 troviamo il tipo di file che adesso è $04 (WBPROJECT = 4).

```
0030: 04000026 FBDE0026 FBCE8000 00008000
0040: 00000000 00000000 00000000 00000000
```

Come negli altri file, troviamo adesso la struttura image, seguita dai due bitmap.

```
0050: 00000028 00150002 00265F96 03000000
0060: 00000000 00000000 3FFFFFCC 00003FFF
0070: FFCF0000 3FFFFFCF C0003803 FFCFF000
0080: 3FFFFFC0 00003803 FFFFFC00 3FFFFFFF
0090: FC003FFF FFFFFC00 3F8400C0 7C003FFF
00A0: FFFFFC00 39008000 7C003FFF FFFFFC00
00B0: 38000040 7C003FFF FFFFFC00 3FFFFFFF
00C0: FC003FFF FE007C00 3FFFFFFF FC003FFF
00D0: FFFFFC00 00000000 00000000 00000000
00E0: FFFFFFFC 0000C000 00330000 C0000030
00F0: C000C000 00303000 C7FC0030 0C00C000
0100: 003FFF00 C7FC0000 0300C000 00000300
0110: C0000000 0300C07B FF3F8300 C0000000
0120: 0300C6FF 7FFF8300 C0000000 0300C7FF
0130: FFBF8300 C0000000 0300C000 00000300
0140: C00001FF 8300C000 00000300 C0000000
0150: 0300FFFF FFFFFF00 00000000 00000000
```

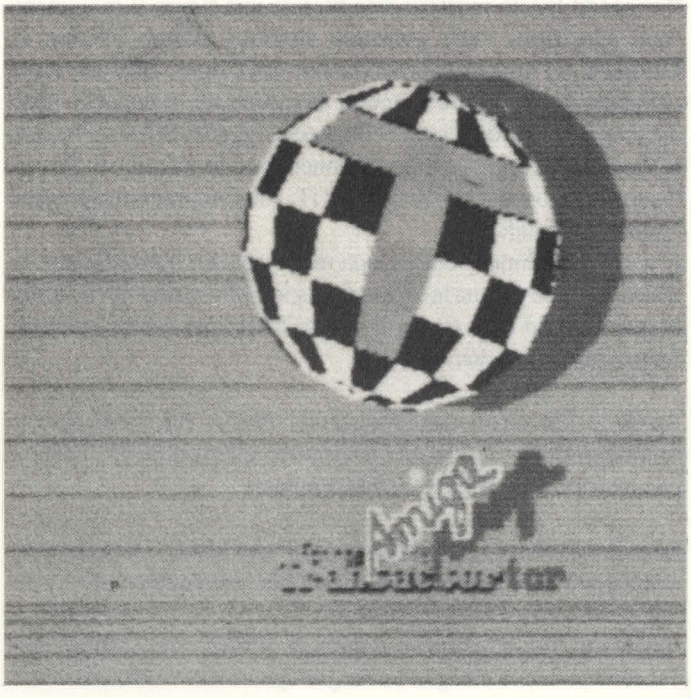
Il tipo Project ha sempre associati un tool di default e anche i "ToolTypes". Questi argomenti, che vengono poi utilizzati dal programma, possono essere modificati dall'utente utilizzando il comando 'Info' che si trova nel menu del Workbench. Le stringhe che rappresentano il nome del file Tool e i "ToolTypes" si trovano in questo punto:

```
0160: 0020576F 726B6265 6E636820 312E323A
0170: 5574696C 69746965 732F4E6F 74657061
0180: 64000000 00100000 00114649 4C455459
0190: 50453D6E 6F746570 61640000 00000E46
01A0: 4F4E543D 746F7061 7A2E2038 00000000
01B0: 1757494E 444F573D 3130302C 2031302C
01C0: 3330302C 31303000
```

Traducendo in ASCII:

```
. Workbench 1.2:Utilities/Notepad
.........FILETYPE=notepad
.....FONT=topaz. 8
.....WINDOW=100, 10, 300,100
```

Ogni stringa è preceduta da una longword che specifica la lunghezza della stringa stessa, incluso il byte a zero posto al termine. Le stringhe che abbiamo appena visto stabiliscono quale font sarà utilizzato, quali dimensioni avrà e quale posizione occuperà la finestra del Notepad quando verrà richiamato attraverso questa icona Project. Il programma Notepad legge questi ToolTypes quando viene attivato e agisce di conseguenza. (Per maggiori informazioni sul meccanismo di funzionamento dei parametri del Workbench e sui ToolTypes, riferitevi all'articolo di Rob Peck apparso sullo scorso numero). Fine della visita. Spero che sia stata utile.



Routine in AmigaBasic per il disegno 3-D

di Antony Bryant

Animazione e visualizzazione di figure tridimensionali

Anthony Bryant è un laureato all' Università di Manitoba. Lavora con sistemi di controllo ed è specializzato in robotica; nel tempo libero programma l'Amiga.

L'AmigaBasic ha un così ricco insieme di comandi che la possibilità di crearne di nuovi è spesso facilitata.
I sottoprogrammi sono la via per espandere il set di comandi disponibili da linguaggio con nuove funzioni utilizzabili in specifiche applicazioni.

La mia collezione di sottoprogrammi, PLOT 3D, è un set di strumenti per il disegno che permette di tracciare punti in uno spazio tridimensinale come definito dai parametri passati a ciascuna funzione.
I sottoprogrammi sono facilmente inseribili alla fine di qualsiasi programma senza aver problemi di ridefinizione di variabili.

Un esempio d'uso di PLOT 3D è dato nel listato 1: il programma mostra quattro viste di una sfera divisa in quadranti come se la si fosse osservata da angoli differenti. Ovviamente, perché il tutto funzioni correttamente è necessario aggiungere alla fine del listato le routine di PLOT 3D.

Il secondo listato mostra come ruotare una figura (i movimenti sono specificati nelle istruzioni DATA) intorno all'asse Z. Cambiando l'angolo di osservazione (ax), ogni volta che viene chiamata la subroutine vengono memorizzate in un array tridimensionale viste differenti dello stesso oggetto. Successivamente l'animazione verrà realizzata ponendo sullo schermo molto velocemente, con un ciclo di GET/PUT, i 30 fotogrammi precedentemente calcolati.
L'array a tre dimensioni occupa molto spazio; il programma ha infatti bisogno di tutta la memoria disponibile residente su Amiga (riferendosi ad un sistema dotato di 512k) per essere eseguito. Spaventati? Sto scherzando!

Partendo dal set di sottoprogrammi PLOT 3D come base di routine grafiche, è possibile espandere ulteriormente l'insieme con nuovi comandi ancora più complessi.

Ecco ora i sottoprogrammi disponibili con PLOT 3D: esaminate attentamente i due listati disponibili per apprenderne l'uso.

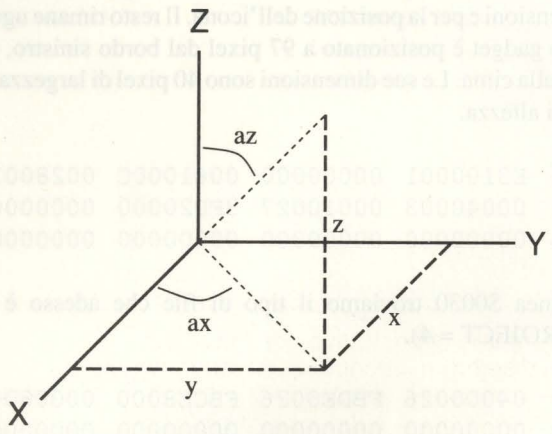
dMoveto (x,y,z) - Muove il cursore grafico (non visibile) alle coordinate date e disegna un singolo punto del colore corrente.

dDrawto (x,y,z) - Disegna una linea dalle coordinate del cursore grafico a quelle passate a questa funzione.

dAngle (ax,az) - Imposta l'angolo di osservazione in gradi (vedi il diagramma).

dParams (x,y,sc,di) - x e y impostano le coordinate dell'origine, sc è il fattore di scala e di è il fattore di scala addizionale per l'uso dell'asse z; 100 e 2.2 sono i valori consigliati per sc e di.

dAxes (xL,yL,zL) - Disegna gli assi di dimensioni date.



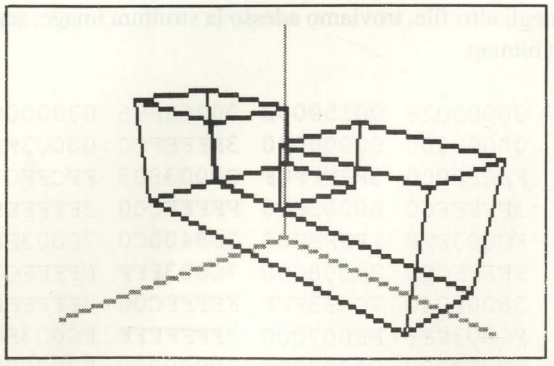
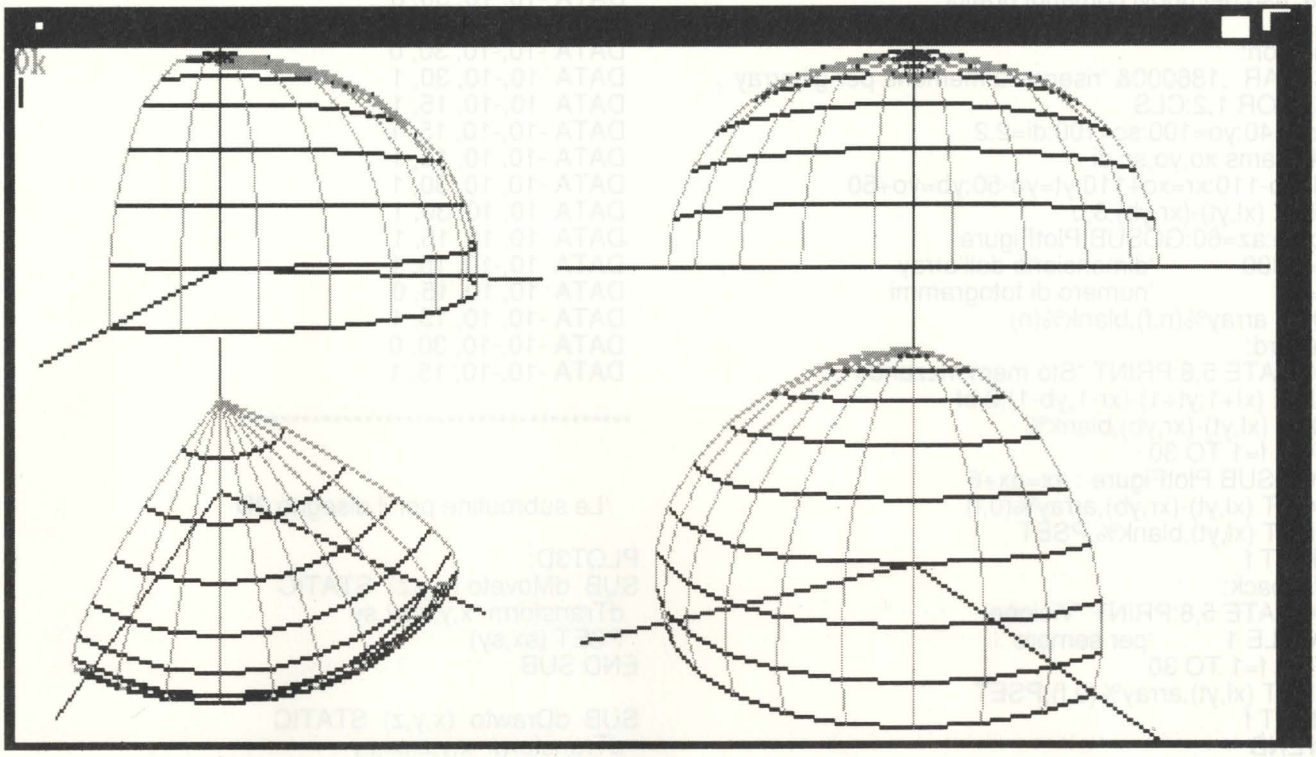


Figura rotante originata dalle routine del secondo listato, ripresa quando ha effettuato solo metà rotazione. Ogni fotogramma nella sequenza di rotazione viene calcolato, disegnato e memorizzato; seccessivamente i fotogrammi memorizzati vengono visualizzati in tempo reale.



Schermo creato dal programma del listato 1: viene disegnata una sfera divisa in quadranti, osservata da punti di vista differenti, utilizzando le routine PLOT 3D.

```
' Listato 1.
' Per AmigaBASIC 512K


' programma dimostrativo per PLOT 3D
' ogni disegno e' ottenuto cambiando angolo di
' osservazione
' Fa' uso dei nuovi comandi grafici


FourViews:
  COLOR 1,2:CLS
  dParams 100!,140!,100!,2.2
  dAngle 15!,80!
  dAxes  75!,75!,75!
  GOSUB PlotQSphere
  dParams 100!,80!,100!,2.2
  dAngle 30!,15!
  dAxes  75!,75!,75!
  GOSUB PlotQSphere
  dParams 440!,140!,100!,2.2
  dAngle 45!,90!
  dAxes  75!,75!,75!
  GOSUB PlotQSphere
  dParams 440!,60!,100!,2.2
  dAngle 60!,60!
  dAxes  75!,75!,75!
  GOSUB PlotQSphere
END

  'la subroutine

PlotQSphere:
  COLOR 1 :r=60 :rad=3.14159/180
  FOR theta=0 TO 90 STEP 10
    z=r*COS(theta*rad): r1=r*SIN(theta*rad)
    x=r1:y=0: dMoveto x,y,z
    FOR beta=0 TO 90 STEP 10
      x=r1*COS(beta*rad): y=r1*SIN(beta*rad)
      dDrawto x,y,z
    NEXT beta
  NEXT theta
  COLOR 3
  FOR theta=0 TO 90 STEP 10
    x=0:y=0:z=r: dMoveto  x,y,z
    FOR beta=0 TO 90 STEP 10
      z=r*COS(beta*rad)
      x=r*SIN(beta*rad)*COS(theta*rad)
      y=r*SIN(beta*rad)*SIN(theta*rad)
      dDrawto x,y,z
    NEXT beta
  NEXT theta
RETURN

 ************************************


' Listato 2.
' Per AmigaBASIC 512k


' rotazione di una figura sull'asse delle Z
' cambiando l'angolo di osservazione "ax"
```

```
' Fa' uso dei nuovi comandi grafici

Rotation:
 CLEAR ,186000& 'riserva la memoria per gli array
 COLOR 1,2:CLS
 xo=140:yo=100:sc=100:di=2.2
 dParams xo,yo,sc,di
 xl=xo-110:xr=xo+110:yt=yo-50:yb=yo+50
 LINE (xl,yt)-(xr,yb),3,b
 ax=0:az=60:GOSUB PlotFigure
 n=2830          'dimensione dell'array
 f=30            'numero di fotogrammi
 DIM array%(n,f),blank%(n)
Record:
 LOCATE 5,8:PRINT "Sto memorizzando ..."
 LINE (xl+1,yt+1)-(xr-1,yb-1),2,bf
 GET (xl,yt)-(xr,yb),blank%
 FOR f=1 TO 30
  GOSUB PlotFigure : ax=ax+6
  GET (xl,yt)-(xr,yb),array%(0,f)
  PUT (xl,yt),blank%,PSET
 NEXT f
Playback:
 LOCATE 5,8:PRINT "Visione          "
 WHILE 1         'per sempre
 FOR f=1 TO 30
  PUT (xl,yt),array%(0,f),PSET
 NEXT f
 WEND
END

  'la subroutine

PlotFigure:
 dAngle ax,az
 dAxes 50!,50!,50!
 COLOR 1
 RESTORE Figuredata
 FOR pnt=1 TO 33
  READ x,y,z,i
  IF i=0 THEN
   dMoveto x,y,z
  ELSE
   dDrawto x,y,z
  END IF
 NEXT pnt
RETURN
 'data per la figura di esempio

Figuredata:
'      x,  y,  z, i (i=0 per moveto)
 DATA -10,-10, 30, 0
 DATA -10,-40, 30, 1
 DATA -10,-40,  0, 1
 DATA -10, 40,  0, 1
 DATA -10, 40, 30, 1
 DATA  10, 40, 30, 1
 DATA  10, 40,  0, 1
 DATA  10,-40,  0, 1
 DATA  10,-40, 30, 1
 DATA  10,-10, 30, 1
 DATA -10,-40, 30, 0
 DATA  10,-40, 30, 1
 DATA -10,-40,  0, 0
 DATA  10,-40,  0, 1
 DATA -10, 40,  0, 0
 DATA  10, 40,  0, 1
 DATA  10, 10, 30, 0
 DATA  10, 40, 30, 1
 DATA -10, 10, 30, 0
 DATA -10, 40, 30, 1
 DATA -10,-10, 30, 0
 DATA  10,-10, 30, 1
 DATA  10,-10, 15, 1
 DATA -10,-10, 15, 1
 DATA -10, 10, 15, 1
 DATA -10, 10, 30, 1
 DATA  10, 10, 30, 1
 DATA  10, 10, 15, 1
 DATA  10,-10, 15, 1
 DATA  10, 10, 15, 0
 DATA -10, 10, 15, 1
 DATA -10,-10, 30, 0
 DATA -10,-10, 15, 1

 ************************************


   'Le subroutine per il disegno 3D

PLOT3D:
SUB dMoveto (x,y,z) STATIC
 dTransform x,y,z,sx,sy
 PSET (sx,sy)
END SUB

SUB dDrawto (x,y,z) STATIC
 dTransform x,y,z,sx,sy
 LINE - (sx,sy)
END SUB

' trasforma un punto in coordinate di schermo
SUB dTransform (x,y,z,sx,sy) STATIC
SHARED s1,c1,s2,c2,c1c2,s1c2,s2c1,s2s1,scx,scy,xo,yo
 xe=-x*s1+y*c1
 ye=-x*c1c2-y*s1c2+z*s2
 ze=-x*s2c1-y*s2s1-z*c2+120
 sx=scx*xe/ze+xo
 sy=199-(scy*ye/ze+yo)
END SUB

' imposta l'angolo di osservazione in radianti
SUB dAngle (ax,az) STATIC
SHARED s1,c1,s2,c2,c1c2,s1c2,s2c1,s2s1
 pi=3.14159: rad=pi/180
 s1=SIN(ax*rad):c1=COS(ax*rad)
 s2=SIN(az*rad):c2=COS(az*rad)
 c1c2=c1*c2:s1c2=s1*c2:s2c1=s2*c1:s2s1=s2*s1
END SUB

' imposta i parametri dello schermo
SUB dParams (x,y,sc,di) STATIC
SHARED xo,yo,scx,scy
xo=x:yo=y:scy=sc:scx=sc*di
END SUB

' disegna gli assi 3D
SUB dAxes (xl,yl,zl) STATIC
 COLOR 0
 dMoveto 0!,0!,0!
 dDrawto 0!,0!,zl
 dMoveto 0!,0!,0!
 dDrawto 0!,yl,0!
 dMoveto 0!,0!,0!
 dDrawto xl,0!,0!
END SUB

 ************************************
```

Breakpoint, parte seconda

di Victor A. Wagner

Prosegue la descrizione delle tecniche di debugging

Vic Wagner cominciò a interessarsi di computer nel 1965 mentre era in servizio di leva, lavorando a uno studio di simulazione di volo digitale dell'aeronautica statunitense. Dal 1966, ritornando un civile, ha lavorato con molti costruttori di minicomputer nell'area del software di base per applicazioni realtime. Nei giorni feriali, dalle 8 alle 5, Vic lavora al Technical Support line (il servizio di supporto telefonico) della Computer Automation Inc., e gestisce 3 sistemi software realtime. La sera e nei fine settimana, passa gran parte del tempo a parlare con Taarna (il suo Amiga 1000), a scrivere programmi e a dialogare su CIS AmigaForum (76046,3004). Vic è considerato un'autorità sul ben noto programma di debugging MetaScope della Metadigm Inc. e recentemente è stato eletto presidente dell'Amiga Friends users' group.

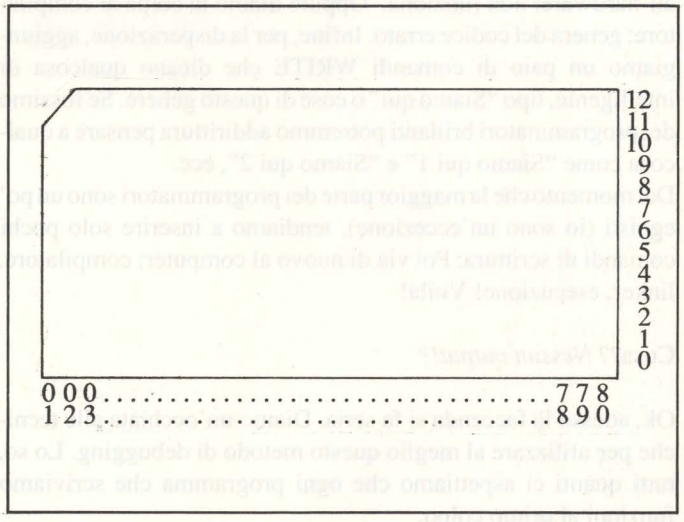
Nella prima parte di questo articolo abbiamo esaminato la storia dei primi computer, fermandoci alla fine degli anni cinquanta-primi anni sessanta (il FORTRAN è stato infatti sviluppato intorno alla metà degli anni cinquanta). Abbiamo visto che la verifica dei programmi era allora pressoché sconosciuta (e spesso lo è ancor oggi).

Le mie attività nel campo dei computer cominciarono nel 1963, quindi, molto di quello che ho raccontato finora lo conosco solamente per sentito dire; d'ora in poi invece, mi riferirò a esperienze personali. Tornando ai tempi passati, sembra che non sia avvenuto molto nel periodo compreso tra i primi anni Cinquanta e il mio inserimento nel mondo dei computer. Il ritmo evolutivo era al tempo piuttosto blando paragonato ai tempi più recenti.

Sono sicuro che starete morendo dalla curiosità di avere la risposta alla domanda con la quale ci siamo lasciati alla fine della prima parte. Fino a ora, tutti i miei amici che hanno letto l'articolo mi hanno guardato, giunti alla fine, dicendo 'PRINT' o 'printf' oppure 'WRITE' o qualsiasi comando per ottenere l'output formattato dal primo HLL (High Level Language = Linguaggio ad Alto Livello) che hanno imparato. Penso che la domanda fosse troppo facile... Questa, infatti, è la risposta giusta.

Prima dell'avvento dei compilatori (o degli interpreti, nel caso del BASIC), era enormemente difficile generare un output da un programma. In realtà non era "difficile", ma raramente avrebbe richiesto un'unica linea di codice sorgente.
Non dimenticate che per i sorgenti dei programmi venivano usati strumenti come nastri e schede di carta. Quelli che utilizzavano le schede erano attratti dal comando WRITE come le api dal miele.

L'idea di inserire una singola scheda nel programma solo per dire "Sono qui!" era preziosa. Ho conosciuto programmatori che avevano mazzi di schede già perforate che utilizzavano per il debugging. Queste schede avevano un colore diverso da quelle normali e generalmente avevano un angolo tagliato.
Cosa? Non sapete cosa intendo quando parlo di schede con gli angoli mancanti? Dunque, per coloro che non sanno come sia fatta una scheda perforata per computer...



...eccone una ricostruzione approssimativa. Le schede erano larghe 80 colonne e alte 12 righe. (Non conosco la ragione per cui avessero 80 colonne, ma questo è probabilmente il motivo per cui ancora oggi abbiamo apparecchi che utilizzano 80 colonne, come i terminali e le stampanti.) Ma l'altezza di dodici righe è forse ancora più difficile da spiegare. Forse basterà dire che Hollerith le progettò così. Nel caso vi chiediate come mai veniva tagliato un angolo, ebbene questo serviva per potere stabilire con una sola occhiata se le schede di un mazzo fossero posizionate correttamente. Se una di queste era girata sotto-sopra o al contrario, ci sarebbe stato un angolo che spuntava. Se le schede per il debugging venivano colorate in arancione o rosso e le restanti erano normali, era più facile trovarle e rimuoverle dal mazzo.

Aprendo una piccola parentesi storica, ecco la ragione per cui le schede perforate erano di quelle particolari dimensioni: Hollerith aveva convinto l'Ufficio del Censimento a costruire una macchina per aiutarli nel loro compito.
Una sera, tardi, un operaio (un meccanico, credo) entrò nell'ufficio di Hollerith e disse che poteva costruire la macchina eccetto

che per un piccolo dettaglio. I disegni non riportavano le misure per le schede che sarebbero state utilizzate, quindi i raccoglitori per queste ultime non potevano essere costruiti fino a quando non fosse stato chiarito quel dettaglio. Hollerith rimase pensieroso per un momento o due, quindi prese dal suo portafoglio un dollaro e lo porse all'operaio dicendo "Fatele di questa misura".
Così le dimensioni standard di una scheda perforata per computer sono le dimensioni della banconota da un dollaro degli inizi del ventesimo secolo.

Riprendendo il nostro discorso, l'idea è quella di ottenere almeno un qualche output dal programma. Quando scrivo e verifico un programma, il periodo più angosciante è il lasso di tempo che precede un output di qualsiasi tipo. Se il nostro programma si limita a partire, a fare lampeggiare una o più spie e quindi ferma il processore (o blocca il sistema operativo), vengono raccolte ben poche informazioni sul suo conto.
Esaminiamo allora attentamente i nostri listati, poi decidiamo invariabilmente che non c'è modo per cui il nostro programma possa non produrre un qualsiasi output. Quindi... diamo la colpa all'hardware: non funziona. Oppure diamo la colpa al compilatore: genera del codice errato. Infine, per la disperazione, aggiungiamo un paio di comandi WRITE che dicano qualcosa di intelligente, tipo "Siamo qui" o cose di questo genere. Se fossimo dei programmatori brillanti potremmo addirittura pensare a qualcosa come "Siamo qui 1" e "Siamo qui 2", ecc.
Dal momento che la maggior parte dei programmatori sono un po' egoisti (io sono un'eccezione), tendiamo a inserire solo pochi comandi di scrittura. Poi via di nuovo al computer; compilatore, linker, esecuzione! Voila!

Cosa?? *Nessun output!?*

Ok, adesso la faccenda si fa seria. Diamo un'occhiata alle tecniche per utilizzare al meglio questo metodo di debugging. Lo so, tutti quanti ci aspettiamo che ogni programma che scriviamo funzioni al primo colpo.
Scommetto che tutti i programmatori sono degli ottimisti. Sembra che non riusciamo mai a inserire nel programma i comandi PRINT se non dopo che il programma si è rifiutato di funzionare. Userò il comando PRINT (il comando di output formattato del BASIC) come rappresentante di tutte le altre istruzioni di output formattato. Questo si basa sull'assunto che tutti i lettori siano familiari almeno con il BASIC.
Ci sono ovviamente alcuni posti in un programma dove risulta spontaneo mettere i comandi PRINT.

1. Prima della chiamata a una subroutine. Questo protrebbe semplicemente consistere in un: PRINT "chiamo xyzzy"

2. Dopo la chiamata a una subroutine. Questo potrebbe essere: PRINT "sono tornato da xyzzy"

3. Prima di un'istruzione IF (magari nella forma IF a<b). Per esempio: PRINT "confronto <",a,b.

Notate che nell'istruzione PRINT vengono stampati anche i valori dei due oggetti che vengono confrontati. Dal momento che possiamo vedere qual'è il più piccolo, possiamo prevedere cosa farà il computer quando raggiungerà l'enunciato IF. Un piccolo avvertimento per quanto riguarda il fatto di stampare dei valori: alcuni linguaggi permettono che le macro o le funzioni abbiano effetti collaterali. Questo significa che l'istruzione PRINT potrebbe non stampare lo stesso valore che verrà invece visto da IF, causandoci molta fatica nel tentativo di stabilire perché il programma non funziona correttamente. Può succedere addirittura che un programma perda alcuni dei suoi input. Per esempio:

```
PRINT "verifico l'uguaglianza",getc(),"="
IF ((c=getc()) == '=')
```

leggerà un carattere, quindi l'istruzione IF ne leggerà un altro. Il programma si comporterà in maniera differente a seconda che sia presente o meno l'istruzione PRINT: questi probabilmente non erano i nostri intenti.

4. Prima di un'istruzione CASE (o SWITCH). Ancora una volta, risulta molto utile stampare anche il valore dell'espressione. Assicuratevi di leggere l'avvertimento contenuto al punto 3.

5. Se il nostro programma crea dei problemi in un ciclo FOR, è utile mettere un comando PRINT che stampi l'indice del ciclo:

```
PRINT "loop 1",i
```

oppure potremmo direttamente aggiungere l'indice del ciclo a un altro comando PRINT eventualmente già esistente. Un paio di cose a cui fare attenzione a questo punto sono:

* mettere un comando PRINT in un ciclo può generare una quantità notevole di output.

* quando esaminiamo l'output, assicuriamoci di controllare il primo e l'ultimo valore dell'indice del ciclo. Questo è il punto dove troveremo quegli stupidi errori di inizio-fine. Sapete ciò che intendo dire: "Oops, questo dovrebbe essere 0 to n-1, non 0 to n".

Un'altra parentesi storica: qualcuno di voi si è mai domandato perché in così tanti esempi di frammenti di programmi si utilizzano, per le variabili indice dei cicli e per i controlli di confine, nomi come: i, j, k, l, m, n? La risposta viene dal linguaggio FORTRAN (che fu il primo HLL disponibile). Nel FORTRAN (nel primo FORTRAN) esistevano due differenti tipi di dati: gli interi e i reali. I tipi delle variabili non venivano dichiarati esplicitamente, come invece avviene nei moderni HLL (successivamente anche compilatori FORTRAN permisero permisero variazioni di questo tipo). A questo punto qualche anima saggia decise che il tipo di una variabile fosse determinato dalla prima lettera che formava il nome della variabile stessa. Le lettere che furono scelte erano: I, J, K, L, M e N (non siete sorpresi, vero?). Perché furono scelte proprio queste lettere dal bel mezzo dell'alfabeto? Ebbene, si tratta, in pratica, delle lettere dalla I alla N e risulta facile ricordare la I e la N perché sono le prime due lettere della parola 'INteger'.

6. Se siamo proprio sicuri che il loop sia corretto, allora potrebbe

essere sufficiente scrivere:

PRINT "finito il loop n"

I punti sopraelencati sono utili per la prima prova del programma e probabilmente non sarà necessario rispettarli tutti. Quello che stiamo facendo, con questa tecnica, è di creare un piccolo programma di controllo sulle istruzioni per un problema specifico. Risulta abbastanza facile esagerare con l'istruzione PRINT. Possiamo riempire un intero scatolone di carta senza compiere quasi alcuno sforzo (vedere la prima puntata per i miei commenti sulle risme di fogli contenenti l'output di un'operazione di tracing). Questa tecnica è stata di moda per parecchi anni. Ricordiamo che, negli anni Cinquanta e nei primi anni Sessanta, il tempo macchina era molto costoso. I programmatori dovevano inoltrare i loro lavori ed esaminare i risultati alle loro scrivanie. C'erano alcune installazioni nelle quali i programmatori addirittura non potevano entrare nella sala del computer. Dovevi passare il tuo mazzo di schede a una persona attraverso una piccola finestra, insieme alle istruzioni su come eseguire il lavoro. Poco tempo dopo potevi ritirare il risultato insieme al mazzo dei sorgenti presso un'altra piccola finestra. Se invece si trattava di una installazione molto grossa, potevi trovare i risultati nel casellario/sul tavolo/nella libreria assegnata al tuo dipartimento.

In alcuni casi il tempo di elaborazione si misurava in giorni. Che cosa potevi fare in attesa che il tuo ultimo programma terminasse? Lavoravi su un altro programma. Oppure miglioravi i dati con i quali verificare il funzionamento del primo programma.

In altre installazioni il tempo macchina veniva suddiviso in ragionevoli intervalli (ragionevoli per l'amministrazione, naturalmente) che venivano distribuiti fra i programmatori in base a qualche algoritmo. Questo algoritmo era sempre così complesso che era impossibile implementarlo in un programma. Bisognava fare richiesta scritta per avere un'ora di tempo macchina. Lo staff amministrativo raccoglieva la tua richiesta insieme a quelle degli altri, quindi faceva qualche operazione magica e in seguito esponeva la tabella. Dovevi farti strada nella folla di programmatori per leggere la tabella appena esposta (e qui una serie di borbottii, mugugni e rumori vari) per scoprire che ti era stata assegnata un'intera ora di tempo macchina domani mattina dalle 0315 alle 0415.

dalle 3:15 alle 4:15 *della mattina!?*

Perché tutte le tabelle di lavoro per il debugging somigliavano a quella? Suppongo che fosse perché i lavori importanti venivano svolti durante il giorno. Lavori come il "resoconto dell'utilizzo del calcolatore" o l'ultima magia del "programma per le tabelle d'uso del calcolatore" che qualche buffone dello staff amministrativo stava tentando di scrivere. Ad ogni modo, eccola lì, un'ora di tempo del computer... tutta per te. Come facevi a sapere che dovevi chiedere un'ora? La maggior parte dei problemi vengono individuati in circa 10 minuti, mentre occorrono parecchie ore per trovare quelli rimanenti. Lo staff amministrativo aveva compilato numerose statistiche sul tempo d'uso e aveva scoperto che il valore medio era di un'ora. Ecco quindi come veniva ripartito il tempo. Adesso esaminavi la tabella un po' più da vicino. Chi ha ottenuto la macchina appena prima di me? Quanto sono bravi? Potrebbero finire prima? Perché non venire un po' prima per dare un'occhiata alle loro faccende? Quest'ultima tattica era veramente una buona idea. Potevi scoprire qualche problema (qualcosa che ti risultava ovvio in quanto non eri rimasto a fissarlo durante le ultime settimane), oppure l'altra persona poteva sentirsi così imbarazzata dal fatto che non riusciva a trovare il problema, da decidere di anticipare la fine del lavoro.

A un certo punto avrebbe detto: "Questo problema mi fa proprio perdere la testa. Penso che mi prenderò un po' di riposo". Dopo avere raccolto tutta la sua roba, ti avrebbe ringraziato per l'assistenza e se ne sarebbe andato.

Poteva anche succedere che la persona prima di te fosse un membro attraente dell'altro sesso e/o comunque una persona divertente... In ogni caso, qualsiasi fosse la ragione, arrivavi prima del tempo. Arrivando, gettavi un'occhiata nella stanza del computer, decidevi quale strategia usare quella notte (per avere prima la macchina), prendevi tutto il materiale necessario (i mazzi di sorgenti, i dati per la verifica e così via) ed entravi deciso nella stanza del calcolatore. C'erano alcuni suoni formali, forse una sorta di formula magica, da dire obbligatoriamente come:

"Non riuscivo a dormire quindi sono venuto prima. Hai bisogno di aiuto?? <sbadiglio>, o forse ti potrei prendere un po' di caffè? <un altro sbadiglio ancora più grande>"

Se in quella stanza del computer era veramente permesso alla gente di bere un caffè, barcollavi in direzione del distributore senza aspettare una risposta. Se eri molto, molto furbo, conoscevi già i gusti dell'altra persona per quanto riguardava lo zucchero (avevi fatto lunghe ricerche a tale proposito). Quando tornavi, ti sedevi, sorbivi un sorso di caffè e usando il tuo migliore tono di voce dicevi: "Come sta andando?"

Non ho ancora incontrato un programmatore che non sia desideroso, anzi, addirittura ansioso, di raccontarmi che problemi sta avendo con questo suo programma intransigente di cui sta facendo il debugging. A questo punto fai attenzione. L'altro programmatore tenterà di interessarti al suo problema; tenterà di farlo sembrare il problema più interessante che tu abbia mai visto. Non c'è niente di male nell'apprendere qualcosa su un programma altrui. Potresti addirittura imparare qualche nuova tecnica di codifica o di debugging. Ascolta attentamente le motivazioni che ti verranno date per sostenere che questo è il problema più fantastico sul quale si sia mai lavorato (potresti averne bisogno, un giorno o l'altro). Mostra di essere coinvolto; in effetti potresti addirittura essere in grado di dare una mano. Comunque, tieni un occhio in maniera discreta sull'orologio, perché sai già come vola il tempo quando sei coinvolto con un computer.

Man mano che il tuo tempo d'uso si avvicina, nota attentamente i commenti e i suggerimenti dell'altra persona. Saranno sicuramente volti a persuaderti a lasciargli usare una parte del tuo tempo per trovare "quest'ultimo bug". Questo momento può essere utilizzato per imparare altre tecniche atte a ottenere con lusinghe

del tempo dalla persona che viene dopo di te. Anche se non troviamo quindi il problema dell'altra persona, c'è comunque l'opportunità di imparare delle tecniche che non vanno accantonate.

Supponiamo di essere finalmente alla macchina, pronti a cominciare le operazioni di debug. Per prima cosa compiliamo/assembliamo/effettuiamo il link del nostro programma, se lo avevamo modificato. Okay, adesso abbiamo un programma eseguibile per la verifica, i listati appropriati, le mappe del link e la reference card del programma di debugging che stiamo per utilizzare. Tutto pronto per cominciare.

Molti computer (se non tutti) possono essere costretti a eseguire a piena velocità un programma e poi a fermarsi in una locazione predefinita, restituendo il controllo a un programma chiamato debugger. In molti casi ciò si ottiene modificando il programma in memoria e sostituendo l'istruzione alla locazione prescelta con una istruzione speciale. In altri casi l'hardware ha la capacità di avvertire un altro programma se si accede a una particolare locazione. Qualunque sia il meccanismo utilizzato, qualche brillante programmatore l'avrà già scoperto e avrà confezionato attorno a questa funzione, chiamata breakpoint, un programma chiamato debugger. Tutti questi programmi permettono di specificare la locazione alla quale iniziare l'esecuzione e almeno una locazione alla quale interromperla. Molti (se non tutti) permetteranno di specificare anche solamente i nuovi punti dove fermarsi e riprenderanno l'esecuzione dall'ultimo punto di interruzione incontrato.

C'è una differenza fondamentale sul come eseguire il debugging a seconda se si dispone del programma in assembly (magari il nostro compilatore HLL produce un listato simile all'assembly) o se si dispone del solo listato HLL. Iniziamo dal primo caso.

Se abbiamo un listato completo in assembly, possiamo 'vedere' quello che dovrebbe esserci in ogni locazione del programma che stiamo esaminando. Potremmo avere bisogno di fare riferimento alla mappa del link e di fare un po' di calcoli (magari il debugger li farà al nostro posto) per essere in grado di determinare dove si trovi esattamente un'istruzione, ma questo non è un problema insormontabile. La maniera nella quale mi comporto in questi casi è questa: leggo il listato dall'inizio del programma simulando le operazioni del computer fino a quando non raggiungo un'istruzione condizionale di qualche tipo. E' qui che posiziono il mio breakpoint. Potrei non avere neanche un'idea di quali saranno i risultati quando avrò raggiunto il breakpoint, ma sarò sicuro che ci arriverò. Faccio quindi partire il programma e ancora prima che abbia il tempo di allontanare le dita dalla tastiera, il breakpoint sarà stato raggiunto. A questo punto esamino i registri del computer.

Molti debugger usati oggi visualizzano automaticamente i registri quando viene raggiunto un breakpoint. Nei primi tempi, invece, l'output avveniva quasi sempre su una periferica lenta, tipo macchina da scrivere, quindi i registri non venivano visualizzati automaticamente. In ogni caso, esaminiamo qualsiasi cosa stia per essere comparata, per sapere in quale modo si comporterà l'istruzione condizionale. Se risulta che il test sta esaminando gli oggetti giusti, ma che i valori di questi non hanno molto senso, dovremo scoprire perché si stiano confrontando i valori sbagliati. Molti debugger ci permettono anche di modificare la memoria, in modo da potere fare dei 'patch' (delle modifiche al codice), in modo che il programma eseguibile funzioni correttamente la prossima volta che verrà utilizzato. Se (come è probabile) ci sono delle variabili non inizializzate, possiamo modificarle ora, magari accedendo al registro che le contiene, prima di continuare il debugging.

La tecnica che utilizza i breakpoint per trovare gli errori in un programma consiste nel mettere una serie di breakpoint nelle locazioni dove siamo ragionevolmente sicuri che il computer arriverà, nell'esaminare i registri e le variabili e nel modificare il codice, ove sia necessario, con dei patch. Se siamo furbi, riporteremo sul listato tutte le modifiche man mano che verranno eseguite, in modo da potere successivamente modificare il sorgente del programma. Quando viene chiamata una subroutine, possiamo decidere se soffermarci anche all'interno di questa, oppure se fermarci alla locazione dove verrà restituito il controllo. La scelta dipende da quanto siamo sicuri che la subroutine non abbia alcun problema.

Occorre fare attenzione a mettere dei breakpoint nei punti d'entrata di una suroutine. La maggior parte delle macchine moderne chiama le subroutine senza modificare le locazioni di entry point (il punto di ingresso di una subroutine), quindi è perfettamente legittimo mettervi un breakpoint. Alcune macchine più primitive (alcune delle quali vengono costruite tutt'oggi) , utilizzano invece un'istruzione che mette l'indirizzo di ritorno dalla subroutine al posto della prima istruzione della subroutine stessa, poi incomincia l'esecuzione dalla seconda istruzione. Questo comando viene generalmente chiamato JST o STJ (Jump and STore, STore and Jump). Se stiamo effettuando il debugging su una di queste macchine, assicuriamoci di non mettere mai un breakpoint esattamente all'ingresso di una subroutine. Se lo dovessimo fare, il breakpoint verrebbe distrutto dall'indirizzo di ritorno da subroutine inserito in quel punto e quindi il debugger non lo incontrerebbe mai.

Se disponiamo unicamente del listato HLL, è necessario applicare un'altra tecnica. In questo caso io personalmente faccio un uso molto intenso della link map. Questo documento dovrebbe elencare tutti i punti di ingresso di tutte le routine globali. Abbiamo anche bisogno di ulteriori informazioni dal produttore del nostro compilatore su come vengono allocati i dati e su come vengono chiamate le subroutine. Occorre sapere dove saranno gli argomenti da passare alle subroutine, dove verranno restituiti i valori delle funzioni e come viene allocato lo spazio per i dati locali. Armati di tutte queste informazioni, potremo applicare con successo questa tecnica di debugging. Invece di fermarci a ogni istruzione condizionale, ci fermeremo solo in corrispondenza dei punti di entrata delle funzioni (e anche ai punti di uscita, ma di questo ne riparleremo più avanti). Dai listati dovremmo essere in grado di individuare una subroutine che il programma deve chiamare, non importa in che modo. Forse ci sarà una lista di svariate subroutine, delle quali una sola verrà effettivamente chiamata. Non importa, mettiamo un breakpoint davanti a tutte. Quando giungiamo all'inizio della subroutine, esaminiamo gli

argomenti che sono stati passati. Se sono corretti possiamo trovare l'indirizzo di ritorno dalla subroutine (dovremo sapere dove viene conservato) o possiamo trovare una subroutine chiamata, a sua volta, da questa. In ciascuno dei due casi non dovrebbe essere molto difficile trovare il punto dove mettere il prossimo breakpoint. Naturalmente, se gli argomenti della subroutine sono sbagliati, dovremo esaminare attentamente il codice tra l'ultimo breakpoint e la chiamata. Questo è il momento in cui serve sapere come viene allocata la memoria per le variabili locali.
Generalmente non è molto sicuro effettuare dei patch su programmi eseguibili scritti in HLL: quindi, una volta trovato l'errore, saremo costretti a cambiare il sorgente, usare il compilatore/linker e proseguire nel debugging.

La possibilità di mettere breakpoint nei programmi ha rappresentato un grosso passo avanti. Si possono infatti correggere manualmente e temporaneamente gli errori, continuando nella verifica dei programmi. Le strategie di ricerca possono essere modificate come risultato del comportamento del programma man mano che se ne manifestano i comportamenti. Mi sono reso conto parecchie volte che avevo sbagliato a interpretare ciò che faceva il programma, per cui i dati che avevo preparato per i test si rivelavano inutili al fine di individuare il problema. Questa era una vera e propria maledizione. I breakpoint sono molto meglio che eseguire una istruzione alla volta (single-stepping), in quanto molte macchine sono in grado di eseguire milioni di istruzioni al secondo, ben più di quante se ne possano eseguire schiacciando un bottone ogni volta. La tecnica generale è simile a quella illustrata negli undici punti contenuti nella prima puntata di questa serie, ma adesso non sussiste più la necessità di darsi da fare per inserire interruzioni o per spostare interruttori hardware. L'intera operazione è controllata dalla tastiera. I programmi si fermano dove vogliamo noi e il contenuto dei registri viene visualizzato per nostra convenienza.

Finalmente adesso siamo riusciti a finirlo! Se solo...

Se solo questo printout non fosse così lento (dieci caratteri al secondo)!... Dove sono andati quei registri? Ho stampato solamente una piccola quantità di memoria e i registri sono stampati qui da qualche parte... Se solo potessi vedere il contenuto dei registri e delle locazioni di memoria mentre il programma è in esecuzione... Quanto fa la somma esadecimale di $A733 e $15FD dove devo mettere il prossimo breakpoint?... Sarebbe bello se il debugger potesse leggere la link map e mi permettesse di inserire dei nomi anziché dei numeri... Hmm, è tempo di ricompilare. Meglio sistemare il sorgente (spero proprio che ci siano delle schede vuote nella perforatrice).
Nella prossima puntata vedremo le rivoluzioni introdotte da:

* I terminali video (output più veloce).
* I file su disco e i DOS (editing/compiling/linking più veloci).
* Le macchine a condivisione di tempo o time-sharing (tutti possono avere una macchina sempre disponibile).
* I personal computer (vantaggi simili al time-sharing).
* Il multi-tasking (è possibile modificare il sorgente senza uscire dal debugger).
* I sistemi operativi a finestre (è possibile vedere più cose contemporaneamente).

SoftMail

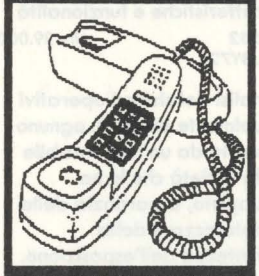
Vendita per corrispondenza di programmi originali ed accessori per computers.

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	22.500
MouseTop	20.000
MouseHouse	20.000
MouseHolder	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
Dungeon master clue	25.000
H-HT: tutti i titoli...	telef.
Might & Magic clue	25.000
Quest for clues	39.000
Sentinel world clue	25.000
Starflight clue	25.000
Wasteland clue	16.500

AMIGA

Aegis draw 2000	telef.
Aegis draw: Art/CAD pack	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
Autoduel	59.000
Barbarian II	telef.
Battlechess	49.000
Black lamp	39.000
Blitzkrieg at Ardennes	59.000
Butcher	59.000
Captain blood	49.000
Carrier command	49.000
Cell animator	telef.
Centerfold squares	59.000
Comic setter	139.000
Comic setter art disks	telef.
Corruption	49.000
Crazy cars	29.000
Def con 5	59.000
Defender of the crown	59.000
Deja vu II	telef.
Deluxe print II	99.000
Deluxe productions	299.000
DigiPaint	99.000
DigiView GOLD	telef.
Director	99.000
Disk drive esterno	299.000
Dos2Dos	75.000
Dragon's lair (1 MB, 6 disk)	telef.
Dragonslayer	telef.
Driller TUTTO italiano	59.000
Dungeon master	telef.
Elite	45.000
Empire	49.000
Express paint 3.0	telef.
Fantavision	125.000
Fire & forget	39.000
Fish	45.000
Flight simulator II	99.000
Scenery disks	telef.
Forms in flight	125.000
Galileo 2.0	99.000
Garfield	49.000
Grabbit	49.000
Hellfire attack	39.000
Hostages	telef.
Impossible mission II	25.000
International soccer	39.000
Italy 90 soccer	39.000
Interchange vari...	telef.
Jet & Japan bundle	110.000
Jinxter	45.000
King of Chicago	49.000
Lattice 5.0	telef.
Leaderboard birdie	29.000
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
Mission: con-bat	telef.
Modeler 3D	129.000
Mortville manor	59.000
Nigel Mansell grand prix	49.000
Obliterator	45.000
Offshore warrior	39.000
Outrun	25.000
Olympic challenge	25.000
Operation cleanstreet	telef.
P.O.W.	59.000
PacMania	25.000
Page flipper Plus	79.000
President is missing	telef.
Ports of call	65.000
ProSound designer	79.000
ProSound w/hardware	199.000
Prowrite 2.0	179.000
Publisher plus	125.000
Reach for the stars	59.000
Return to Genesis	39.000
Rocket Ranger	59.000
SEUCK	telef.
S.D.I.	59.000
Sculpt/Animate 4D	telef.
Sentinel	49.000
Shadowgate	49.000
Sinbad	59.000
Skyfox II	59.000
Speedball	49.000
Star Goose	39.000
Star ray	49.000
Starfleet	59.000
Starglider II	49.000
Strike force harrier	49.000
The bard's tale II	49.000
The works!	249.000
Thexder	59.000
Three Stooges	59.000
Time & magic	45.000
Times of Lore	telef.
Tracker	49.000
TV Sports: Football	telef.
TV*Show	125.000
TV*Text	125.000
Ultima IV	49.000
Universal military sim.	45.000
Data disks 1 e 2	telef.
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
Who framed Roger Rabbit	telef.
World Class Leaderboard	telef.
Write & file	125.000
Zing keys	25.000
Zoetrope	telef.
Zoom	45.000
3 Demon	145.000
4x4 Off road racing	25.000

COMMODORE 64 CASS.

Around the world in 80	25.000
Barbarian II	18.000
Dark side	18.000
Driller	25.000
Flight simulator II	39.000
Game over I+II	18.000
Gold Silver Bronze	18.000
Hawkeye	18.000
Impossible mission II	12.000
Last Ninja II	25.000
MicroSoccer	telef.
Red storm rising	39.000
Shoot'em up constr. kit	25.000

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
Captain blood	29.000
Carrier force	55.000
Caveman ugh olympic	telef.
Chessmaster 2100	69.000
Deathlord	35.000
Driller	29.000
Echelon (con LipStick)	59.000
Eternal dagger	35.000
Faery Tale adventure	telef.
Fast break	29.000
Game over I+II	25.000
GEOS 2.0	95.000
GeoPublisher	100.000
Grand Prix Circuit	telef.
Gulf strike	49.000
Home video producer	69.000
Infocomics vari...	telef.
Mars saga	35.000
MicroSoccer	telef.
Neuromancer	39.000
Panzer strike	45.000
Red storm rising	49.000
Rocket Ranger	telef.
Shoot'em up constr. kit	35.000
Stealth mission	75.000
The bard's tale III	35.000
Ultima V	49.000
Wasteland	39.000
Who framed Roger Rabbit	telef.
Zak Mc Kracken	39.000

COMMODORE 128 DISCO (128K, 80 COL.)

"C" Language	99.000
Basic 8.0	75.000
GEOS 128	telef.
Mach 128	125.000
Superpack II	179.000

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 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) AMJ

Cognome e nome _____

Indirizzo _____ Nr. _____

CAP _____ Città _____ Prov. _____ Telefono _____

FIRMA (SE MINORENNE QUELLA DI UN GENITORE)

VERRANNO EVASI SOLO GLI ORDINI FIRMATI _____

AT 2

Come internazionalizzare i vostri programmi

di John Toebes

L'arte di creare delle applicazioni curate

John Toebes è un analista senior di sistemi presso il SAS Institute Inc. a Cary nel Nord Carolina, ed è l'autore del compilatore Lattice C V4.0. Programma con l'Amiga dal 1985 ed è coordinatore per la The Software Distillery presso la quale è responsabile di molti popolari programmi di pubblico dominio come Hack, PopCLI, Memwatch e Blink. Può essere contattato tramite Bix (JTOEBES) e tramite il BBS della Software Distillery al numero (919) 471-6436.

La potenza e la flessibilità dell'Amiga l'hanno reso una macchina internazionale presente sul mercato con una grande varietà di configurazioni differenti. Nonostante una schiera di utenti così differenziata, esistono dei trucchi estremamente semplici che rendono più facile cooperare con degli ambienti così diversi.

Ci sono tre aree di influenza basilari, spesso non molto considerate dai programmatori, che richiedono delle attenzioni speciali quando ci si rivolge a un'utenza internazionale:

1) Le configurazioni del video.
2) Le configurazioni della tastiera.
3) Le lingue nazionali.

Con una piccola quantità di lavoro i primi due sono in realtà dei problemi inesistenti. L'ultimo è relativo a un'area dove si può lasciare spaziare la propria creatività. Vediamo cosa è possibile fare.

Primo giorno - Vedere i segni.

A causa della sua capacità di supportare gli standard video è possibile che l'Amiga abbia successo come macchina per il desktop video. Infatti è distribuita con due diversi tipi di chip video: l'NTSC per gli Stati Uniti e il PAL per i paesi europei. Su di un Amiga con lo standard NTSC, la risoluzione dello schermo è di 200 linee per 320 o 640 pixel, mentre su di un Amiga con lo standard PAL si hanno 256 linee, sempre per 320 o 640 pixel. Un programma che sia limitato alle proporzioni NTSC coprirà solamente l'80% dello schemo quando verrà eseguito su di un Amiga PAL. Al contrario, eseguendo un programma specifico per il PAL su di una macchina NTSC si perderà il 20% dello schermo.

Non si possono mettere da parte le differenze sui video asserendo semplicemente che un programma verrà eseguito con uno o l'altro dei due standard, perché esistono dei fattori che possono causare delle differenze anche nell'ambito di un singolo standard. Per esempio, 'MoreRows' è un programma che permette all'utente generico di aumentare la risoluzione dello schermo del Workbench. Questo articolo è stato scritto su uno schermo del Workbench con la risoluzione di 670 per 460, anche se sfortunatamente il mio editor favorito - mg1b - non riconosce le linee e le colonne in più che esso offre. Inoltre si deve considerare l'interlacciamento, che raddoppia l'altezza effettiva dello schermo. Ultimamente, sono stati realizzati nuovi monitor (il 2024, per esempio) che hanno la possibilità di visualizzare delle risoluzioni maggiori. E voi non comprereste sicuramente un programma che utilizzi solamente una piccola porzione del vostro nuovo monitor da 1024 per 800 pixel.

Per mettere d'accordo tutte queste eventualità, ci sono poche e semplici regole da seguire:

1) Non codificate *mai* in modo rigido i limiti dimensionali di uno schemo o di una finestra. Un buon esempio di come 'non' comportarsi lo si trova nel secondo esempio in "Intuition: The AMIGA User Interface":

```
struct NewScreen NewScreen = {
        0,      /* Il margine sinistro deve essere
                   uguale a 0 */
        0,      /* Margine superiore */
        320,    /* Ampiezza (bassa risoluzione) */
        200,    /* Altezza (non interlacciato */
....
NewWindow.MaxWidth = 640;
NewWindow.MaxHeight = 200;
```

Bisogna invece usare i valori forniti in GfxBase per calcolare il corretto numero di righe e colonne.

```
/* Calcola la dimensione per una finestra considerando
* anche il modo interlacciato e qualsiasi modo con
* più righe
*/
rows=GfxBase->NormalDisplayRows;
cols=GfxBase->NormalDisplayColumns;
GetPrefs(&prefs,sizeof(struct Preferences));
if(prefs.LaceWB)
        rows+=rows;
```

Sebbene io lo sconsigli, si può controllare se si è in standard NTSC oppure PAL guardando direttamente in GfxBase->DisplayFlag e testando i bit di NTSC, PAL o GENLOCK. Fidandosi di questi bit, come indicatori di dimensione dello schermo, non si gestiscono però programmi come 'MoreRows' oppure nuovi monitor con dimensioni maggiori.

Questo richiama l'attenzione su quello che io considero uno dei modi migliori di imparare come lavora l'Amiga. Spendete un attimo del vostro tempo per scrivere dei semplici programmi di test che esplorino le strutture dati del sistema e che usino delle semplici chiamate del sistema.

2) Progettate la disposizione dello schermo in modo *'algoritmico'*. Non date per scontato che un gadget debba essere posizionato con un dato offset per essere piazzato sul fondo dello schermo. Usate GRELWIDTH, GRELBOTTOM, GRELRIGHT e GRELHEIGHT e fate in modo che sia il sistema a farlo per voi. Vedremo più avanti altre cose riguardo a questo argomento.

3) Siate preparati a gestire degli schermi molto ampi. Al crescere delle dimensioni degli schermi la memoria si restringe. Nonostante non sia strettamente in relazione al tipo di display, questa è spesso la causa del fatto che dei programmi che ben si adattano a macchine NTSC non riescono a entrare in macchine PAL.

Uno dei maggiori problemi dell'avere a che fare con display video variabili (in particolare PAL) è che non esiste alcun modo di controllare le differenti configurazioni senza avere a disposizione sia un Amiga NTSC, sia un Amiga PAL. Effettivamente, si può controllare la conformità allo standard PAL su di una macchina NTSC abbastanza facilmente. Invece di aprire uno schermo con dimensioni 640/200, lo si apre come uno schermo 640/256 interlacciato. Sebbene lo schermo non copra l'intero display, e ci possa essere un po' di sfarfallio, questo metodo può dare la possibilità di vedere come appaia la disposizione dello schermo in questa situazione. Per testare la conformità allo standard NTSC su di una macchina PAL non c'è bisogno di usare il modo interlacciato, basta aprire uno schermo alto 200 pixel.

Secondo giorno - Capire il linguaggio.

Un programma, a differenza di ciò che riguarda il display, deve essere più attento per accorgersi delle differenze di tastiere. Ogni paese ha la sua tastiera, mappata in modo diverso, con i tasti differenziati sia come posizione che come codici. Persino negli Stati Uniti esistono tre diverse disposizioni per la tastiera, inclusa quella di Dvorak conosciuta come USA2.

L'Amiga permette di ricevere un input in quattro modi diversi:

1) Leggendo da una finestra CON: o RAW: (Da notare che lo stdin, in un programma C, punta tipicamente qui).

2) Leggendo da un "console.device" assegnato a una window.

3) VANILLAKEYS di Intuition.

4) RAWKEYS di Intuition.

Di tutti questi modi, solamente RAWKEYS può potenzialmente presentare problemi legati alla tastiera; gli altri interpretano automaticamente i tasti battuti in modo appropriato. Nell'ultimo caso, bisogna interpretare i tasti secondo la mappa corrente di default. Per realizzare ciò, la V 1.2 fornisce una nuova routine, RawKeyConvert, che effettua la conversione, includendo la gestione dei "dead keys" (che sono provvisti di accenti). La routine è descritta in dettaglio nel manuale "Amiga Enhancer Software" allegato con la versione aggiornata.

Per usarla, si deve prendere il messaggio da RAWKEYS e trasformarlo in un evento di input. Lo si passa insieme a un buffer, che serve a contenere il risultato della conversione, alla routine RawKeyConvert; si passa anche un puntatore alla mappa di tastiera che si vuole usare. Se si passa NULL come puntatore alla KeyMap, il sistema ne sceglierà una di default. Da notare che bisogna aprire il "console.device" prima di usare la routine. Una tipica applicazione potrebbe essere il codice seguente:

```
struct Device *ConsoleDevice;
struct IOStdReq devreq;
char keybuf[10];

OpenDevice("console.device", -1, &devreq, 0);
ConsoleDevice = devreq.io_Device;
...
for(;;)
{
/* Attende che si verifiche il successico evento */
while ((message = (struct IntuiMessage *)
  GetMsg(window->UserPort)) == NULL)
  Wait(1 << window->UserPort->mp_SigBit);

  switch(message->Class)
  {
  ...
        case RAWKEY:
              len =KeyConvert(message,keybuf,
              sizeof(keybuf));
              /* ora i tasti convertiti si trovano in keybuf
              */
  ...

/*****************************************************************/
/* Converte un messaggio RAWKEY in un testo in input
                                              usando la */
/* keymap di default per la tastiera                    */
/* Parametri:    msg     - Messaggio RAWKEY da
                                          Intuition a chi non  */
/*si deve aver ancora risposto.                         */
/*buf,len - Buffer (con lunghezza) che deve              */
/*contenere i tasti convertiti.                         */
/*****************************************************************/
int KeyConvert(msg,buf,len)
struct IntuiMessage *msg;
char *buf;
int len;
{
        static struct InputEvent ievent = {
              NULL, IECLASS_RAWKEY, 0, 0, 0
        };
        if (msg->Class != RAWKEY)
```

```
        return(0);

    /* Converte l'evento Intuition RAWKEY
                                in un evento Input */
    ievent.ie_code = msg->Code;
    ievent.ie_Qualifier = msg->Qualifier;
    ievent.ie_position.ie_addr = *(APTR *)
    msg->IAddress;

    return (RawKeyConvert(&ievent, buf, len, NULL);
}
```

Una volta ottenuti i caratteri in input, non bisogna commettere l'errore di considerali tutti con codice minore di 127. Alcuni programmi, che provengono da macchine basate su UNIX, mascherano il bit più significativo con istruzioni del tipo:

```
c &= 127;
```

Sull'Amiga una cosa del genere è un disastro perché normalmente vengono usati tutti i 256 caratteri. La maggior parte dei caratteri internazionali hanno il bit di ordine alto posto a 1, in modo da generare alcuni simboli come il copyright e il trademark registrato. In genere, bisogna acquisire ciò che l'utente scrive in modo letterale. I tentativi di miglioramento, trasformando l'input in lettere maiuscole o con la lettera iniziale maiuscola, spesso si trasformano in disastri.

Terzo giorno - Parlare la lingua.

L'ultima parte da considerare per realizzare un programma curato su Amiga è quella che riguarda la gestione delle varie lingue disponibili. Sfortunatamente, relativamente a questo problema, non esiste nessun prontuario che renda più semplice la vita. Il miglior approccio possibile è di essere flessibili. Oltre al fatto che non tutti al mondo parlano inglese, a parecchie persone semplicemente non piace il modo in cui vengono espressi certi termini, oppure vogliono solamente poter personalizzare il programma.

Esistono come minimo tre tipi diversi di approccio al problema:

1) Scrivere tutto in inglese e prevedere che chiunque voglia utilizzare il programma con una lingua diversa modifichi tutti i messaggi utilizzando un'utility. Questa sarebbe una buona soluzione se tutte le lingue avessero parole della stessa lunghezza o comunque più compatte dell'inglese. Lingue come il tedesco sono spesso più estese, e ciò rende il metodo improponibile. Un altro punto a sfavore è che non renderebbe possibile creare delle disposizioni dello schermo basate sulla lunghezza dei messaggi.

2) Eliminare totalmente qualsiasi messaggio testuale. è l'estremo opposto e fornisce un'opportunità eccellente per creare una completa ambiguità. Il detto che 'un disegno è meglio di mille parole' è vero quando si ha una piccola icona per rappresentare un'operazione.
Il problema è che l'utente sia in grado di discriminare quale, tra le mille parole, sia quella che si voleva che lui capisse. Per illustrare il problema, pensate a tutti i segnali stradali che date come scontati, solo perché siete stati istruiti a riconoscerli.

Questo non significa che non si debbano usare delle icone o delle immagini, solamente non bisogna contare su di una raffigurazione grafica per comunicare ogni tipo di concetto. La grafica è generalmente più utile alla rappresentazione di operazioni che bisogna svolgere piuttosto che alla descrizione dei risultati di qualche attività.

3) Permettere all'utente di configurare i messaggi del programma. Il modo migliore è di creare un file di configurazione dei messaggi (preferibilmente associato con il programma oppure nella directory s:) e di leggerlo all'inizio del programma. In questo modo, l'output e le stringhe di prompt sono resi completamente flessibili e viene fornita l'opportunità, anche alle persone che parlano la stessa lingua, di personalizzare completamente il programma secondo i propri gusti.

Sebbene questa terza soluzione possa sembrare abbastanza difficile da realizzare, lo è solamente per la prima stringa. Una volta che si riesce a stabilire un meccanismo per memorizzare e accedere al messaggio che bisogna stampare, aggiungere le altre stringhe è molto semplice. Quando si escogita qualcosa per un programma, in genere poi funzionerà sulla maggior parte degli altri programmi. Il programma 'chatty.c' riportato più avanti illustra questo tipo di implementazione.

Quarto giorno - Uscire per incontrare i nativi.

Solo per il fatto che si abbia una stringa da stampare non è detto che il lavoro sia finito. Intuition 1.3 include diversi nuovi font di caratteri, compresi i font Times proporzionali e ciò rende la vita un po' più interessante. Non si può più pensare semplicemente alla disposizione dello schermo in termini di celle di caratteri, si deve invece programmare lo schermo in termini di attributi dei font.

Intuition fornisce, per gestire i font, diverse routine per determinare la dimensione del testo. Il modo più semplice è di usare IntuiTextLenght. La si può chiamare con un puntatore a una struttura IntuiText (esattamente quella che si deve usare per creare un gadget o un elemento di un menù). Bisogna però seguire un ben preciso procedimento. La serie di caratteri a cui si fà riferimento (se ne esiste una) deve risiedere in memoria. Se invece è presente su di un disco, bisogna prima chiamare la routine OpenDiskFont per caricarla in memoria. Una struttura TextAttr definisce il font desiderato; un puntatore a questa struttura sarà presente nella struttura IntuiText, ed è passato a OpenDiskFont.

IntuiTextLenght restituisce l'ampiezza del testo in pixel. Per determinarne l'altezza, si deve usare il campo YSize nella struttura del font che si è aperta. In questo modo, per trovare l'area che la scritta 'Box Me' occuperebbe, usando il font sapphire/11, bisogna utilizzare un programma del tipo:

```
struct TextAttr mytextattr;
struct TextFont *font=NULL;
static struct IntuiText itext={
        0,1,JAM1,0,0,NULL,NULL,NULL
```

```
};

    DiskfontBase=OpenLibrary("diskfont.library",0);

    mytextattr.ta_Name="sapphire.font";
    mytextattr.ta_YSize=11;
    mytextattr.ta_Style=0;
    mytextAttr.ta_Flags=0;

    font=OpenDiskFont(&mytextattr);

    itext.ITextFont=&mytextattr;
    itext.IText="Box Me";

    width=IntuiTextLenght(&itext);
    height=font->tf_YSize;

    CloseFont(font);
    CloseLibrary(DiskfontBase);
```

Naturalmente, si devono anche controllare i problemi relativi all'apertura dei vari file senza dare nulla per scontato. Non bisogna neanche dimenticare l'estensione .font del nome passato alla routine OpenDiskFont.

Quinto giorno - Mettere tutto insieme.

Una volta capite tutte le problematiche implicate, si può passare a un programma pratico, utile per impadronirsi meglio della tecnica. Il programma di esempio, Chatty.c apre semplicemente una finestra ampia come l'intero schermo del Workbench e stampa su di essa dei messaggi casuali estratti da una lista predefinita. Inoltre fornisce un gadget di tipo booleano che permette di scegliere tra una frequenza veloce/lenta, di stampa dei messaggi. Sebbene un programma del genere sia effettivamente molto semplice, permetterà di evidenziare molti degli argomenti visti finora.

1) Gestisce il modo video interlacciato, *MoreRows* e NTSC/PAL; può lavorare infatti con schermi di qualsiasi dimensione.

2) I messaggi che vengono scritti sono personalizzabili dall'utente finale senza dover ricompilare il programma.

3) Può lavorare in multitask - non genera dei busy wait neanche ciclando su di un loop di stampa.

4) Quando viene disattivato, termina senza recare danni al sistema.

5) Il gadget che viene creato si dimensiona automaticamente in funzione della dimensione della finestra che viene selezionata.

6) Ha consapevolezza del tipo di font e di quali siano le sue dimensioni.

Come è stato detto precedentemente, il problema di gestire le dimensioni dello schermo è facilmente risolvibile facendo riferimento alle informazioni contenute in GfxBase e in Preferences. Sono state usate queste informazioni per aprire una finestra ampia quanto lo schermo del Workbench. Se vengono utilizzati altri monitor o se si modificano le dimensioni di default dello schermo, il programma si adatta automaticamente alla nuova situazione, cosa che potrebbe essere interessante per un programma di gestione di testi.

Gestire la visualizzazione del testo è forse la parte più difficile di tutto il programma, poiché bisogna riuscire a scrivere delle stringhe in modo che l'utente le possa personalizzare, senza dover ricompilare il programma. Per realizzare questa opportunità le stringhe sono state memorizzate in un file esterno. Bisogna comunque stare attenti a dove posizionarle, poiché devono risiedere in una zona strettamente associata al programma stesso. Inoltre il programma deve avere la capacità di funzionare anche senza questi messaggi.

Esistono tre zone ovvie dove è possibile memorizzare i messaggi:

1) In un file che risieda nella stessa directory del programma.

2) In un file posto nella directory S:

3) In un file che abbia l'estensione .info associato al programma.

Le scelte 1 e 2 in realtà non sono molto efficenti perché presuppongono che l'utente ricordi l'associazione esistente tra i file, mentre la terza ha il vantaggio che gli utenti sono già abituati a copiare il file .info (il Workbench copia questo file in modo automatico). Quindi la scelta più logica è di utilizzare questo tipo di approccio.

Il Workbench fornisce un'ottima zona, nel file .info, dove è possibile memorizzare le informazioni: il vettore di TOOL TYPES *(per ulteriori informazioni consultate l'articolo di Rob Peck "Riflessioni sul Workbench", apparso nel numero precedente)*. Per vederlo bisogna semplicemente selezionare un qualsiasi tool e fare apparire la finestra di informazione del Workbench. In questa finestra è presente, in basso, un rettangolo denominato TOOL TYPES con due frecce per scorrerlo e con due gadget, ADD e DEL, usati per creare e per cancellare degli elementi. Per aggiungere un tool, bisogna selezionare ADD e inserire le informazioni relative nel gadget che riceve la stringa. Le stringhe esistenti possono essere modificate posizionandosi su un dato tool con le frecce e selezionando il gadget che permette di inserire la nuova stringa. Un elemento di questo vettore è molto simile a una istruzione di assegnamento del tipo:

NAME=text

Con questa convenzione, si possono semplicemente scegliere dei nomi per le opzioni che vogliamo personalizzare e accedere a essi attraverso il programma. Nel programma Chatty vengono riconosciuti i seguenti tool:

OPENFAIL = messaggio da stampare quando non sia possibile aprire una finestra.

TITLE = titolo della finestra che bisogna aprire.

FONT = nome del font di caratteri da usare.

FONTSIZE = dimensione del font da usare (un intero).

FONTSTYLE = intero che descrive lo stile. (1=UNDERLINED 2=BOLD 4=ITALIC 8=EXTENDED)

FONTFLAGS = intero per i flag del font. (2=REVERSE 32=PROPORTIONAL)

GADGET = titolo per il gadget che controlla la velocità.

STRINGA = stringa qualsiasi.

...

STRINGZ = stringa qualsiasi.

Se una particolare opzione non viene specificata, Chatty sceglierà un default appropriato e, se il file .info mancasse addirittura, il programma sarebbe ancora in grado di funzionare. è importante che venga preso un accorgimento di questo tipo quando si progetta un programma che può essere configurato. Un utente è sempre libero di usare il Workbench per modificare le stringhe desiderate.

E' molto semplice accedere alle stringhe contenute nel file .info, dal punto di vista del programma. Lo si realizza in tre fasi:

1) Apertura della icon.library in IconBase, in modo da poter usare le routine del Workbench che gestiscono le icone.

2) Localizzazione del file .info. A causa della presenza sia del CLI che del Workbench si hanno due diverse situazioni da prendere in considerazione.

a) Sotto CLI, il nome del programma è contenuto in argv[0]. Questo dato, unitamente alla directory corrente che è memorizzata nella struttura del processo, indica il punto da cui si è partiti.

b) Dal Workbench, si riceve un messaggio di startup che indica il nome del programma e della directory dalla quale si è partiti. Usando CurrentDir ci si può spostare in quella directory e richiedere al Workbench il DiskObject associato al file, il quale è una copia, residente in memoria, del contenuto del file .info. Dal DiskObject si può risalire al vettore TOOL TYPES e quindi a tutte le stringhe inserite dall'utente.

3) Per ciascuna stringa, alla quale siamo interessati, si effettua una chiamata alla funzione FindToolType() per cercare il vettore TOOL TYPES. è stata creata una routine getstr() per gestire i valori di default quando un'opzione non è stata specificata. La getstr() riceve come parametri il tool da cercare e un default appropriato. Per esempio, il titolo della finestra è derivato da:

```
getstr(toolptr,"TITLE","No title specified");
```

Se getstr() non riesce a trovare TITLE nel vettore, ritornerà la stringa di default che sarà quella che useremo. Questo tipo di procedura garantisce di avere sempre qualcosa con cui lavorare.

I messaggi possono anche essere gestiti in modo più trasparente usando una macro del tipo:

```
#define M_CANTOPEN (getstr(toolptr,"OPENFAIL","Can't
        open the window"))
```

la quale permette di scrivere:

```
puts(M_CANTOPEN);
```

che funziona anche quando non esiste un messaggio specificato dall'utente.

Chatty usa pochi messaggi che quindi possono essere associati a dei nomi descrittivi. In programmi di dimensioni maggiori che debbano trattare molti messaggi, è meglio usare dei numeri di messaggio e stampare il numero associato a un dato messaggio, come default, quando la stringa non è presente.

Notate che non si è imposto alcun limite ai messaggi. Il programma permette all'utente di specificare il nome del font di caratteri, la loro dimensione e i flag associati, il titolo della finestra e il testo per il gadget di controllo della velocità. Questa tecnica può essere facilmente utilizzata per realizzare dei menù e anche dei requester, tenendo conto che l'unico limite è dato dalla dimensione massima del vettore TOOL TYPES (32K). Un programma che supera questo limite dovrebbe probabilmente appoggiarsi a metodi alternativi di configurazione (è difficile che un utente voglia usare la finestra Info, per modificare il vettore, quando esistono così tante stringhe).

I problemi relativi a un corretto multitasking e una disattivazione automatica sono gestiti attraverso lo stesso meccanismo. Con questo tipo di programma dimostrativo, l'approccio tipico è quello di creare un ciclo continuo che controlli spesso l'esistenza di un messaggio. Questo comporta che a volte un operazione venga eseguita troppe volte, o che si perda tempo aspettando che accada qualcosa, sebbene non si possa parlare di un busy form look in senso stretto.

Un'alternativa è quella di usare il timer in modo da regolare le operazioni, aspettando un determinato intervallo di tempo tra una e l'altra. Esiste però un modo più semplice per ottenere un funzionamento di questo tipo. Se si imposta il flag INTUITICKS IDCMP nella definizione della finestra, si otterrà un flusso costante (circa 30 al secondo) di messaggi di evento. La stabilità di questa frequenza non è così accurata da permettere delle temporizzazioni critiche ma consente al programma di eseguire del lavoro 'ogni tanto'.

Per quanto riguarda il nostro programma, lo schermo viene aggiornato solo quando viene ricevuto un messaggio. Usando INTUITICKS si ha un flusso di lavoro costante senza degrado delle prestazioni. Un altro vantaggio che ne deriva è che quando

la finestra non è selezionata il programma non riceve più messaggi, il che permette all'applicazione di disattivarsi.

Il gadget di controllo della velocità implementato nel programma evidenzia come sia possibile creare un gadget che si modifica automaticamente in funzione della dimensione della finestra. In questo caso si è semplicemente voluto che occupasse il bordo inferiore (escluso il gadget di dimensionamento) e a questo proposito si sono usati i flag GRELBOTTOM e GRELWIDTH e sono stati inseriti dei valori negativi nei campi 'top' e 'width'. Il valore del campo 'top' indica quanto distante sia il fondo della finestra dal punto iniziale del gadget, mentre il campo 'width' indica quanto spazio lasciare quando si ridimensiona il gadget.

L'ultima cosa illustata dal programma è come determinare la dimensione di una stringa. Quando si stampano le stringhe, il programma determina una posizione di stampa in modo casuale, prima calcolando l'area nella quale possa entrare la stringa (usando il codice visto precedentemente) e poi cercando un posto nella finestra dove piazzarla. In questo modo, a meno che la stringa non sia più ampia che l'intera finestra, non si dovrebbe mai scrivere sopra i bordi.

Sesto giorno - Far rotta verso casa.

Come si può notare analizzando questo piccolo esempio, gestire i problemi che un programma ben scritto comporta non richiede molto codice. I trucchi e le subroutine che sono stati presentati possono essere facilmente trasportati su di un altro programma che sia destinato a viaggiare per il mondo. Quello che conta veramente è di essere a conoscenza dei problemi.

Come nota finale, non ho gestito deliberatamente in questo programma un esempio di internazionalizzazione. Controllate ciò che avete appreso cercando e risolvendo questo problema.

Soluzione:

(non leggete oltre se volete raccogliere la sfida!)

L'altezza del gadget dovrebbe essere derivata dal tipo di font scelto dall'utente. Per default, il testo del gadget è stampato usando il font di sistema, ma non è stato preso alcun provvedimento per controllare il particolare font richiesto e neppure per cercare il font di default di sistema. Bisogna solo aggiungere le seguenti linee:

```
speedtext.ITextFont = &mytextattr;
speed.Height = fontheight;
speed.TopEdge = -fontheight;
fontheight += fontheight-8;
/* Considera dello spazio in più sopra il gadget di
dimensionamento */
```

subito dopo la linea che dice:

```
fontheight = font->ft_YSize;
```

```
/*****************************************************************/
/* CHATTY.C - Programma dimostrativo per illustrare la
                                                    scrittura   */
/*di programmi internazionali                                    */
/* Di John A. Toebes, VIII                                       */
/* Basato sull'originale pubblicato su:                          */
/*The Amigan Apprentice and Journeyman Volume III,
                                                    Number 2    */
/*  P.O. Box 411, Hatteras NC 27943                             */

*****************************************************************/
#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/gfxbase.h>
#include <workbench/workbench.h>
#include <workbench/startup.h>
#include <workbench/icon.h>
#include <libraries/dosextens.h>
#ifdef MANX
#include <functions.h>
#define U_ARGS(a) ()  /* Niente supporto per i prototypes
                                                delle funzioni*/
#else
#define U_ARGS(a) a   /* prototyping delle funzioni
                                                attivato */
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/icon.h>
#include <proto/dos.h>
#define textAttr TextAttr
#include <proto/diskfont.h>
#include <string.h>
#include <dos.h>
#endif

#define BASEROW 11  /* Prima riga sulla quale possiamo
                                                scrivere */
#define BASECOL 4   /* Prima colonna sulla quale
                                        possiamo scrivere */
#define EXTRAROW 20 /* Overhead per title bar, borders
                                        e sizing gadget */
#define EXTRACOL 8  /* Overhead per i border */

#define MAXSTRINGS  26
extern struct WBStartup *WBenchMsg;
struct IntuitionBase *IntuitionBase;
struct Library *DiskfontBase;
struct GfxBase *GfxBase;
struct Library *IconBase;
struct IntuiText speedtext = {
  1, 0, JAM1, 5, 0, NULL, "Go Fast", NULL};

struct Gadget speed = {
 NULL, 4, -8, -20, 8,  /* Next, left, top(v), width(v), height */

 GADGHCOMP|GRELBOTTOM|GRELWIDTH,
 TOGGLESELECT, /* flags, activation */
 BOOLGADGET, NULL, NULL, &speedtext, 0, /*
  type, render, select, text, mutex */
 NULL, 0, NULL };  /* specialinfo, gadgetID, Userdata */

struct NewWindow newwindow = {
 0, 0, 0, 0, /* left, top, width, height - filled in later */
 -1, -1,     /* Detail, block pens */
 NEWSIZE|CLOSEWINDOW|INTUITICKS, /*
 IDCMP flags */
 WINDOWSIZING|WINDOWDEPTH|WINDOWCLOSE|
 WINDOWDRAG|
```

```
  SIMPLE_REFRESH|ACTIVATE,
  &speed, NULL, NULL, NULL, NULL, /* gadgets,
  checkmark, title,screen, bitmap */
  30, 30, 99999, 99999, WBENCHSCREEN /* min/max
  width/height, type */
};

struct IntuiText itext = {
  0, 1, JAM1, 0, 0, NULL, NULL, NULL };

struct DiskObject *diskobj;
struct Library *DiskFontBase;
char *getstr U_ARGS((char **, char *, char *));
char **findtools U_ARGS((char *, BPTR));

#define M_CANTOPEN (getstr(toolptr, "OPENFAIL",
  "Non posso aprire la window"))

#define MSG(a) Write(Output(),a,strlen(a))

void main(argc, argv)
int argc;
char **argv;
{
struct Preferences prefs;
short len, rows, cols, x, y;
struct Window *window;
struct IntuiMessage *message;
static short modes[4] = { JAM1, JAM2, COMPLEMENT,
  INVERSVID };
char *(strings[MAXSTRINGS]);
int count, i;
struct WBArg *wbarg;
char **toolptr, *name, lookup[8];
struct TextAttr mytextattr;
struct TextFont *font = NULL;
int fontheight;

  /* Open all of our libraries that we need */
  if ((IntuitionBase = (struct IntuitionBase *)
    OpenLibrary("intuition.library",0)) == NULL)
  {  MSG("intuition.library?");  goto done;  }

  if ((GfxBase = (struct GfxBase *)
    OpenLibrary("graphics.library",0)) == NULL)
  {  MSG("graphics.library?");  goto done;  }

  if ((IconBase = OpenLibrary("icon.library",0)) ==
        NULL)
  {  MSG("icon.library?");  goto done;  }

 /* Non importa se questa fallisce. Semplicemente non
                                  cambieremo il font */

  DiskfontBase = OpenLibrary("diskfont.library", 0);

  GetPrefs(&prefs, sizeof(struct Preferences));

  /* Vediamo dove siamo stati lanciati e recuperiamo la
                                  nostra icona */
  if (argc) /* CLI */
    toolptr = findtools(argv[0],
        ((struct Process *)FindTask(NULL))-
        >pr_CurrentDir);
  else /* Workbench */
  {
  wbarg = &(WBenchMsg->sm_ArgList[WBenchMsg-
        >sm_NumArgs-1]);
      toolptr  =  findtools(wbarg->wa_Name,  wbarg-
>wa_Lock);
    }
    /* Vediamo se ci hanno dato il nome per la finestra */
    newwindow.Title = getstr(toolptr, "TITLE",  "No ICON file
for TITLE");

   mytextattr.ta_Name  = getstr(toolptr, "FONT",    "topaz");
    mytextattr.ta_YSize = atoi(getstr(toolptr,
                                    "FONTSIZE",  "9"));
    mytextattr.ta_Style = atoi(getstr(toolptr,
                                  "FONTSTYLE", "0"));
    mytextattr.ta_Flags = atoi(getstr(toolptr,
                                  "FONTFLAGS", "0"));

    speedtext.IText = getstr(toolptr, "GADGET",
                                        "Piu' veloce");
   if (DiskfontBase != NULL) /* assicuriamoci di potere usare
                                        i font */
      font = OpenDiskFont(&mytextattr);

    if (font != NULL)
    {
      itext.ITextFont = &mytextattr;
      fontheight = font->tf_YSize;
    }
    else
      fontheight = 9;  /* assumiamo qualcosa di ragionevole
*/

    /* Adesso prendiamo le stringhe per la tabella */
    strcpy(lookup, "STRINGA");
    count = 0;
    for (i = 0; i < MAXSTRINGS; i++)
    {
      if ((name = FindToolType(toolptr, lookup)) != NULL)
                              strings[count++] = name;
      /* Avanziamo fino alla prossima stringa */
      lookup[6]++;
    }

    /* Quando non abbiamo stringhe utilizziamo qualcosa di
                                        ovvio */

    if (count == 0)
      strings[count++] = "Non trovo le stringhe con i
                                        messaggi";


    /* Calcoliamo quanto grande possiamo aprire la
                                  finestra, tenendo conto */
    /* dell'interlace e di eventuali altre specifiche fornite */
    rows = GfxBase->NormalDisplayRows;
    cols = GfxBase->NormalDisplayColumns;
    if (prefs.LaceWB)
      rows += rows;
    newwindow.Width = cols;
    newwindow.Height = rows;

    if ((window = OpenWindow(&newwindow)) == NULL)
    {
      MSG(M_CANTOPEN);
      goto done;
    }
    /* calcoliamo l'overhead causato dal border */
    rows -= EXTRAROW + fontheight;
    cols -= EXTRACOL;

    for(;;)
    {
      /* Attendiamo che accada il prossimo evento */
```

```
    while ((message = (struct
        IntuiMessage *)GetMsg(window->
        UserPort)) == NULL)
      Wait(1 << window->UserPort->mp_SigBit);

    switch(message->Class)
    {
      case CLOSEWINDOW:  goto done; /* Usciamo! */


      case NEWSIZE:      cols = window->Width-
                                        EXTRACOL;
                         rows = window->Height-
                              (EXTRAROW+fontheight);
                         break;

      case INTUITICKS:  break;  /* questo ci permette
                                   di andare lenti */
    }

    /* facciamo sapere a Intuition che abbiamo processato
                                        il messaggio */
    ReplyMsg((struct Message *)message);


    /* vediamo quante volte dobbiamo farlo */
    i = (speed.Flags & SELECTED) ? 5 : 1;

    while(i—)
    {
      /* prepariamo le stringhe da visualizzare (random) */

      itext.IText = strings[((rand() >> 2) % count)];
      itext.FrontPen = ((rand() >> 2) & 3);
      itext.BackPen  = ((rand() >> 2) & 3);
      itext.DrawMode = modes[((rand() >> 2) & 3)];


      /* vediamo dove possiamo metterle con sicurezza */

      len = IntuiTextLength(&itext);
      x = BASECOL + ((rand() >> 2) % (cols - len));
      y = BASEROW + ((rand() >> 2) % rows);

      /* visualizziamo le stringhe */
      PrintIText(window->RPort, &itext, x, y);
    }
  }
done:
  /* Clean up generale */

  if (font)
    { CloseFont(font);                font      = NULL;
}

  if (DiskfontBase);
    { CloseLibrary(DiskfontBase);
                                DiskfontBase  = NULL; }

  if (diskobj)
    { FreeDiskObject(diskobj);
                                diskobj       = NULL; }

  if (window)
    { CloseWindow(window);
                                window        = NULL; }

  if (GfxBase)
    { CloseLibrary((struct Library *)GfxBase);
                                GfxBase       = NULL; }

  if (IntuitionBase)
    { CloseLibrary((struct Library *)IntuitionBase);
                                IntuitionBase = NULL; }

  if (IconBase)
    { CloseLibrary(IconBase);   IconBase      = NULL;   }
}



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Subroutine: findtools                                               */
/* Compito:   individuare un array tools per un dato
                                                        nome            */
/* Parametri:  name - nome del file da cercare
                                        (terminato con zero)            */
/*           lock - directory nella quale cercare il file
                                                        richiesto       */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


char **findtools(name, lock)
char *name;
BPTR lock;
{
BPTR olddir;
char **tools = NULL;

  if (lock == NULL) return(NULL);
  olddir = CurrentDir(lock);
  if ((diskobj = GetDiskObject(name)) != NULL)
    tools = diskobj->do_ToolTypes;
  CurrentDir(olddir);
  return(tools);
}



/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/* Subroutine: getstr                                                  */
/* Compito:   determinare una stringa per un certo
                                                        handle          */
/* Parameters: toolptr - puntatore all'array di
                                                        tooltypes       */
/*          name - nome dell'handle da cercare                         */

/*           deflt - stringa da restituire se handle non viene
                                                        trovato         */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


char *getstr(toolptr, name, deflt)
char **toolptr, *name, *deflt;
{
char *found;

  found = FindToolType(toolptr, name);
  return(found ? found : deflt);
}


#ifndef MANX

/* Trucco per rendere il codice più piccolo */
void MemCleanup(){}
                                                        #endif
```

Principi di Ray Tracing

di Cathryn E.F. Graham

La teoria sulle immagini magiche

Cathryn Graham è nata a Calcutta in India ed è stata educata in Scozia. Dopo aver lavorato alla Cambrigde University con suo marito Eric, si sono trasferiti entrambi sulle montagne del New Mexico per sviluppare software a tempo pieno. Cathryn si è specializzata nella documentazione del software e ha scritto diversi manuali di software inclusi quelli di ***Sculpt-3D*** *e* ***Animate-3D****, programmi scritti da Eric. Sta attualmente lavorando sul progetto e sul test di un nuovo sistema per la creazione di modelli per solidi sull'Amiga.*

Il ray tracing è uno dei modi più efficaci per realizzare immagini realistiche di vedute tridimensionali. Se avete visto immagini, generate da un computer, di oggetti solidi che mostravano ombre, zone evidenziate, riflessioni e trasparenze allora probabilmente sono state realizzate con la tecnica del ray tracing.

L'Amiga è l'unico computer di basso costo che permette la visualizzazione contemporanea sullo schermo di un grande numero di colori. Il modo HAM (hold and modify) può visualizzare fino a 4096 colori simultaneamente e ciò è abbastanza per realizzare immagini realistiche. Sono stati scritti un certo numero di programmi di ray tracing per l'Amiga, tra i quali il programma Dbw-Render, Ssg, che è stato usato per generare le immagini di The Juggler, Sculpt 3D, Silver e Animate 3D.

La rappresentazione di oggetti

Prima di poter pensare a come creare delle immagini, bisogna fornire al computer una rappresentazione degli oggetti tridimensionali che realizzano la scena. A causa del fatto che una figura complicata, come un elefante, è difficile da descrivere matematicamente, è molto più facile ricavare oggetti complicati da altri più semplici. Esistono due approcci differenti al problema: si può costruire un oggetto partendo da volumi solidi o da superfici. Il volume definibile matematicamente nel modo più semplice è la sfera, poiché può essere descritta in termini relativi alla posizione del suo centro e del suo raggio. è quasi altrettanto facile lavorare con altre figure, come blocchi rettangolari, cilindri, coni ecc., ma a differenza della sfera bisogna anche tener conto dell'orientamento nello spazio dell'oggetto. Si possono combinare volumi solidi in diversi modi differenti. Due oggetti possono essere sommati, o combinati per produrne un terzo. Oppure un oggetto può essere "sottratto" da un altro; per esempio se si sottrae un cilindro da un cubo, è come aver creato un foro nel cubo. Un terzo modo di combinare due volumi è chiamato "intersezione". L'intersezione di due volumi è un nuovo volume che consiste nella regione comune ai due volumi originari.

La tecnica di costruire oggetti da volumi diversi è chiamata Geometria Costruttiva Solida (Gcs). Questa tecnica è eccellente nella costruzione di modelli di oggetti realizzati in fabbrica, perché operazioni come tagliare, fresare, perforare e ruotare possono essere facilmente rappresentate. I programmi per Amiga Dbw-Render, Ssg e Silver adoperano tutti la Gcs.

Un inconveniente con la Gcs è che, mentre è molto valida per realizzare un modello di una mensola o di un'altra figura regolare, non lo è altrettanto per quanto riguarda la creazione di oggetti più irregolari. Chi è un potenziale creatore di modelli di elefanti o è interessato alla costruzione di oggetti complicati spesso descrive le sue scene in termini di rappresentazione di superfici. La superficie è costituita mettendo insieme facce triangolari, quadrilatere o poligonali.

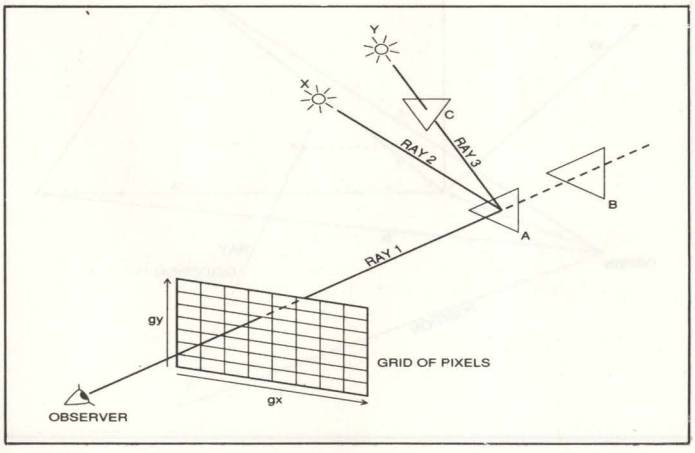


Figura 1: Geometria per il ray tracing.

Le facce possono essere sia curve che piane. Se si usano facce curve, allora le si può fondere insieme, senza linee di giunzione, per la rappresentazione di oggetti curvi. L'uso delle facce piane è invece riservato a immagini che abbiano un aspetto sfaccettato. Se si realizzano delle facce abbastanza piccole si ha una maggiore approssimazione con una curva levigata. Come vedremo più avanti, esiste un trucco che può essere usato per far sì che le facce piane appaiano curve. Sculpt 3D e Animate 3D usano, per la creazione dei modelli, delle facce piane.

Ray tracing

Come suggerisce il nome stesso, la tecnica del ray tracing lavora seguendo i raggi di luce quando rimbalzano attraverso uno

scenario. Se si usano, in modo appropriato, le leggi fisiche della riflessione e dell'illuminamento l'immagine della scena che ne deriva può avere un notevole realismo.

Oltre a specificare tutti gli oggetti che sono presenti nel nostro ambiente simulato dobbiamo anche definire un osservatore. Bisogna considerare l'osservatore come un occhio (vedi Figura 1) o come a una telecamera. In entrambi i casi l'osservatore possiede un certo numero di proprietà :

1) Una posizione relativa agli altri oggetti presenti nel mondo.

2) La direzione di osservazione. Può essere descritta come una coppia di angoli che rappresentano l'azimut e l'altitudine. Oppure la direzione di osservazione può essere specificata fornendo una punto in mezzo agli oggetti, e stabilendo che questo punto sia anche il centro dell'immagine.

3) Un angolo di inclinazione. Come ben sanno i fotografi, la maggior parte delle fotografie ben realizzate sono prese con la macchina a livello con l'orizzonte. Se si immagina che l'osservatore sia a bordo di un aereo sarebbe preferibile avere la macchina fotografica puntata lungo l'aereo anche se così non sarebbe a livello con l'orizzonte.

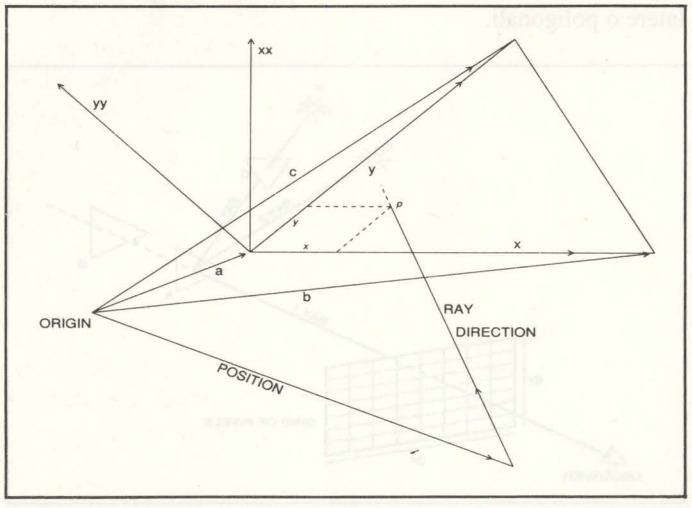


Figura 2: Vettori usati per calcolare l'intersezione tra un raggio e un oggetto triangolare.

4) Un controllo di zoom. Bisogna specificare in qualche modo un angolo di veduta, per esempio fornendo la lunghezza focale della lente attraverso la quale l'osservatore sta guardando. Questo può essere usato per modificare l'effetto della prospettiva. Un angolo di veduta molto ampio può far sembrare la prospettiva esagerata.

La maggior parte dei raggi di luce che rimbalzano attraverso la scena non raggiungono l'osservatore e non contribuiscono alla formazione dell'immagine. Il modo più semplice per selezionare i raggi che effettivamente servono è quello di seguire la luce al contrario partendo dalla posizione dell'osservatore.

Ciò è rappresentato in Figura 1. Si immagini che davanti all'osservatore sia posta una griglia di pixel. Ciascun quadratino della griglia corrisponde a un pixel nell'immagine finale. Il colore e la luminosità da associare a ogni pixel dell'immagine finale è ciò che l'osservatore vedrebbe guardando attraverso la griglia verso la scena. Si traccia una linea, chiamata raggio, dall'osservatore nella direzione di un particolare pixel. In Figura 1, RAY1 ha una intersezione con i due oggetti A e B. A meno che l'oggetto A non sia trasparente possiamo benissimo tralasciare l'oggetto B poiché sarà nascosto alla vista. La luminosità del pixel sarà la luminosità dell'oggetto A nel punto dove è stato colpito dal raggio.

La luminosità di una superficie dipende da due fattori. Il primo sono le proprietà di riflessione della superficie stessa. Se la superficie non è lucida riflette sono una frazione fissa della luce, avente un particolare colore, che la colpisce. Se usiamo un sistema basato su tre colori con una luminosità dei pixel differenziata per la luce rossa, verde e blu, le proprietà di riflessione di una superficie possono essere descritte fornendo tre numeri che rappresentano la frazione di luce riflessa per ogni colore. Per esempio i valori [0.95,0.90,0.05] rappresentano un oggetto di colore tendente al giallo mentre [0.21,0.18,0.23] uno grigio scuro.

Il secondo fattore che condiziona la luminosità di una superficie è la quantità e il colore della luce che vi incide. Ciò può essere calcolato costruendo raggi dalla superficie a ciascuna delle sorgenti luminose che illuminano la scena. Per ciascun raggio si può vedere se interseca con altri oggetti. In Figura 1, RAY3 colpisce l'oggetto C sul percorso verso la sorgente Y, mentre il percorso del RAY2 verso la sorgente X non è ostacolato da nessun oggetto. Se l'oggetto C è opaco non bisogna considerare nessun contributo all'illuminazione da parte della sorgente Y, mentre quello che deriva dalla sorgente X è inversamente proporzionale al quadrato della distanza tra la sorgente X e l'oggetto A.

Un altro fattore che influenza il modo in cui una sorgente di luce illumina un oggetto è l'angolo con cui essa incide sull'oggetto stesso. Lavorando in tre dimensioni è conveniente sfruttare una direzione che formi un angolo retto con la superficie; questa direzione è detta 'normale alla superficie'. L'illuminazione è attenuata da un fattore proporzionale al coseno dell'angolo tra la luce incidente e la normale alla superficie.

Pro e contro

Questa semplice descrizione del comportamento della luce può essere usata per realizzare immagini notevolmente realistiche, complete di ombre e sfumature. Per tutta la nostra spiegazione sul ray tracing si è supposto che ciascun oggetto sia opaco e che rifletta la luce in modo uniforme in tutte le direzioni. Non è però difficile aggiungere altre caratteristiche a questo modello.

Se l'oggetto A fosse semi-trasparente allora l'oggetto B sarebbe parzialmente visibile. L'algoritmo standard del ray tracing può essere applicato al raggio che arriva da dietro l'oggetto A per calcolare il contributo alla luminosità da parte dell'oggetto B. Se l'oggetto A rappresentasse la superficie di qualcosa fatto di vetro allora il raggio potrebbe essere deflesso in modo da rappresentare l'effetto della rifrazione.

In modo analogo, se A fosse una superficie a specchio si potrebbe tenere conto di un raggio riflesso e applicare nuovamente l'algoritmo di ray tracing. Le superfici reali hanno spesso delle proprietà riflettenti molto complicate.
Generalmente la riflessione è aumentata di molto quando l'angolo tra il raggio diretto verso l'osservatore e la componente riflessa del raggio illuminante è piccolo. Questo da luogo allo 'scintilliò che si può osservare con oggetti che, sebbene non siano specchi, sono leggermente lucidi.

E' molto facile aggiungere effetti ottici addizionali a un programma di ray tracing.
Poche dozzine di linee di codice possono aggiungere un nuovo effetto fisico, così che è difficile evitare di apportare continui miglioramenti al codice.
Questo ha portato alla definizione di un cliché nei confronti delle immagini generate col ray tracing. Troppo spesso si sono prodotte immagini di sfere riflettenti che oscillano su delle scacchiere, mentre la tecnica del ray tracing è in grado di generare molte delle scene immaginabili.

Matematica del ray tracing

Un programma di ray tracing deve essere in grado di eseguire tutti i tipi di calcolo geometrico in tre dimensioni.
I ricordi della geometria e dei metodi di analisi imparati a scuola, e le difficoltà affrontate nel calcolare anche solo le proprietà di semplici triangoli bidimensionali, renderebbero molto scoraggiante accostarsi alla matematica del ray tracing. Fortunatamente esiste un modo migliore per eseguire dei calcoli geometrici detto geometria vettoriale.

Un vettore è un'astrazione matematica che possiede sia una dimensione che una direzione.
Uno spostamento fisico può essere rappresentato da un vettore, per esempio 'muoviti 10 metri verso nord'.
Per indicare una posizione in uno spazio tridimensionale si userà un sistema di coordinate. Ciascuna posizione verrà identificata da un insieme di tre numeri e tutte le misure verranno fatte a partire da un singolo punto detto origine. I numeri rappresentano le distanze dall'origine lungo tre direzioni che sono tra di loro ortogonali, per esempio direzione verso est, verso nord e verso l'alto.

Una volta specificata un'origine ogni punto dello spazio può essere indicato con un vettore che rappresenta la variazione di posizione del punto rispetto all'origine. Queste variazioni si possono accumulare e quindi i vettori si possono sommare tra di loro. La seguente funzione, scritta in C, realizza una somma tra vettori.

```
vecadd(a,b,c)      /* somma di vettori a=b+c */
double a[3],b[3],c[3];
{
 int i;
 for(i=0;i<3;++i)
   a[i]=b[i]+c[1];
}
```

I vettori possono anche essere sottratti e copiati:

```
vecsub(a,b,c)   /* sottrazione di vettori a=b-c */
double a[3],b[3],c[3];
{
 int i;
 for(i=0;i<3;++i)
   a[i]=b[i]-c[i];
}


veccopy(a,b)      /* copia di vettori a=b */
double a[3],b[3];
{
 int i;
 for(i=0;i<3;++i)
   a[i]=b[i];
}
```

Nel linguaggio dei vettori un numero normale è spesso definito scalare. Uno scalare può essere moltiplicato per un vettore e il risultato è ancora un vettore:

```
vecsprod(a,s,c)          /* prodotto scalare tra vettore e
                                                  numero */
double a[3],s,c[3]; /* a=s*c */
{
 int i;
 for(i=0;i<3;++i)
   a[i]=s*c[i];
}
```

Due vettori possono essere moltiplicati tra di loro e danno come risultato uno scalare; questa operazione è chiamata prodotto scalare. Il codice per realizzarla è molto semplice:

```
double dot(a,b)   /* Prodotto scalare tra vettori */
double a[3],b[3];
{
 return a[0]*b[0]+a[1]*b[1]+a[2]*b[2];
}
```

Da notare che il prodotto scalare tra un vettore e se stesso non è altro che la sua dimensione elevata al quadrato. Un vettore la cui lunghezza è uguale a uno è anche chiamato vettore unitario e può essere visto in termini di sola direzione. Un vettore normale può essere trasformato in un vettore unitario nel modo seguente:

```
vecsprod(x,1.0/sqrt(dot(x,x)),x);
```

Il prodotto scalare ha una sua interpretazione geometrica: esso rappresenta il prodotto delle lunghezze dei vettori e del coseno dell'angolo compreso tra i vettori stessi. In particolare, se i vettori

sono perpendicolari allora il loro prodotto scalare è nullo.

Esiste un terzo modo con cui si possono moltiplicare dei vettori fra di loro ed è chiamato prodotto vettoriale perché il risultato della moltiplicazione è ancora un vettore.

```
vecprod(a,b,c)  /* prodotto vettoriale a=b^c */
double a[3],b[3],c[3];
{
 double t0,t1;
 t0=b[1]*c[2]-b[2]*c[1];
 t1=b[2]*c[0]-b[0]*c[2];
 a[2]=b[0]*c[1]-b[1]*c[0];
 a[1]=t1;
 a[0]=t0;
}
```

Il prodotto vettoriale ha l'interessante proprietà che il vettore risultante è perpendicolare a entrambi gli altri vettori. La sua dimensione è uguale al prodotto tra le dimensioni dei vettori e il seno dell'angolo compreso tra i vettori stessi.

Ora conosciamo abbastanza strumenti per il trattamento dei vettori da essere in grado di descrivere quasi ogni tipo di problema geometrico; in particolare possiamo costruire dei raggi. Un raggio non è altro che una linea che partendo da un particolare punto si dirige in una particolare direzione data e può essere rappresentato con la seguente struttura in C:

```
struct ray{
  double position[3],direction[3];
};
```

Un determinato punto lungo il raggio può essere calcolato con la seguente funzione:

```
raypos(point,t,ray)
double point[3],t; struct ray *ray;
{
   vecsprod(point,t,ray->direction);
   vecadd(point,point,ray->position);
}
```

A volte una descrizione di questo tipo è detta definizione in forma parametrica perché la variabile 't' è chiamata parametro. A valori differenti di 't' corrispondono posizioni differenti nello spazio, lungo il raggio.

Costruiamo ora un raggio simile a RAY1 in Figura 1 che corrisponde, per esempio, a una particolare posizione (i,j) di un pixel. Il codice relativo è il seguente:

```
double viewdir[3];
/* vettore unitario che indica la direzione
                                dello sguardo dell'osservatore */
double obspos[3];          /* posizione dell'osservatore */
double gx[3],gy[3];        /* vettori che rappresentano
                                la griglia dei pixel */
double d;                  /* distanza dell'osservatore
                                dalla griglia */
int i,j;                   /* pixel considerato */
int nx,ny;                 /* numero di pixel nelle
                                due dimensioni */
struct ray ray;            /* raggio da calcolare */
double temp[3];            /* vettore temporaneo */

.... /* inizializzazione delle variabili */
vecsprod(ray.direction,d,viewdir);
vecsprod(temp,i/((double)nx-1)-0.5,gx);
vecadd(ray.direction,ray.direction,temp);
vecsprod(temp,0.5-j/((double)ny-1),gy);
vecadd(ray.direction,ray.direction,temp);
veccopy(ray.position,obspos);
```

Il raggio che abbiamo calcolato parte dal punto di osservazione e passa attraverso il punto specificato sulla griglia dei pixel. Il prossimo passo è di determinare se il raggio ha delle intersezioni con un particolare oggetto. Se gli oggetti sono dei triangoli possono essere rappresentati dalla posizione dei loro tre vertici:

```
struct triangle {
   double a[3],b[3],c[3],color[3];
} tri;
```

Dividiamo ora il problema in due parti. Per prima cosa bisogna trovare il punto p dove il raggio colpisce il piano che contiene il triangolo. In seguito si controllerà se p appartiene al triangolo. Se si calcolano due vettori X e Y paralleli ai lati ba e ca del triangolo allora si può usare il loro prodotto vettoriale per determinare la normale unitaria n alla superficie del triangolo (vedi Figura 2).

```
double p[3],n[3],x,X[3],y,Y[3],t,pminusa[3];
vecsub(X,tri.b,tri.a);
vecsub(Y,tri.c,tri.a);
vecprod(n,X,Y);
/* lo si rende un vettore unitario */
vecsprod(n,1.0/sqrt(dot(n,n)),n);
```

Se il punto p è sul piano del triangolo allora il vettore (p-a) è perpendicolare a n, cioè:

$$(p-a)\bullet n=0$$

dove '•' indica un prodotto scalare. Se il punto p è anche lungo il raggio allora :

$$p=position+t*direction$$

dove l'asterisco è usato per indicare la moltiplicazione fra un parametro scalare t e il vettore che rappresenta la direzione del

raggio. Si devono combinare ora le ultime due equazioni e calcolare il valore di t che corrisponde al punto sul raggio che sia anche sul piano del triangolo.

```
vecsub(pminusa,ray.position,tri.a);
t=-dot(pminusa,n)/dot(ray.direction,n);
raypos(p,t,&ray);
```

Ora il punto p è sul piano del triangolo ma bisogna determinare se appartiene al triangolo stesso. Per verificarlo bisogna esprimere p in termini di due vettori X e Y che siano paralleli a due lati del triangolo.

```
p=a+x*X+y*Y
```

Così p è descritto in termini di coordinate (x,y) in un sistema di coordinate oblique i cui assi sono X e Y, ma X e Y probabilmente non sono perpendicolari. Calcoliamo due altri vettori XX e YY entrambi nel piano del triangolo, ma con XX perpendicolare a X e YY perpendicolare a Y. Questi vettori possono essere ricavati nel modo seguente:

```
vecprod(XX,n,X);
vecprod(YY,n,Y);
```

Calcolando il prodotto scalare dell'espressione di p con XX e YY è possibile ricavare i valori di x e y.

```
vecsub(pminusa,p,tri.a);
x=dot(pminusa,YY)/dot(X,YY);
y=dot(pminusa,XX)/dot(Y,XX);
```

Infine il punto p è contenuto nel triangolo solo se x e y sono positivi e x+y è minore di 0.5; si può quindi restituire il risultato di un test su di una intersezione in un modo simile a questo:

```
return (x>0.0) && (y>0.0) && (x+y<0.5);
```

Nei programmi di ray tracing reali ciascun raggio deve essere testato in riferimento a ogni oggetto e l'oggetto più vicino è quello che ha il parametro t con il valore positivo più piccolo.
Il passo successivo è di calcolare la luminosità di un oggetto nel punto p. Una fonte di luce potrebbe essere definita dalla seguente struttura:

```
struct lamp {
  double position[3],brightness[3];
} lamp;
```

Il vettore dall'oggetto alla sorgente luminosa e la distanza tra i due possono essere calcolati con il seguente codice:

```
double lvec[3],distance;
vecsub(lvec,lamp.position,p);
distance=sqrt(dot(lvec,lvec));
```

Sebbene sia stata calcolata la normale unitaria dell'oggetto triangolare non è noto da quale parte del triangolo emerga la normale. Se il triangolo è una parte di un volume solido è buona norma fare in modo che la normale sia direzionata in modo da puntare verso l'esterno dell'oggetto. Oppure si può decidere che la normale sia diretta verso l'osservatore cambiandone il verso, se necessario, come segue:

```
if(dot(ray.direction,n)>0.0)
  vecsprod(n,-1.0,n);
```

Se il vettore verso la sorgente è diretto nella direzione opposta della normale allora quella sorgente sta illuminando la parte dell'oggetto che non è di fronte all'osservatore e quindi non contribuisce all'illuminazione dell'immagine in quel punto, altrimenti si può calcolare il contributo all'illuminazione per un pixel:

```
double pixbrite[3],cosine;
int k;
cosine=dot(lvec,n)/distance;
for(k=0;k<3;++k)   /* loop tra rosso, verde e blu */
 if(cosine>0.0)
  pixbrite[k]=lamp.brightness[k]
  *tri.color[k]*cosine/
  (distance*distance);
 else
  pixbrite[k]=0.0;
```

Si devono calcolare i contributi da parte di ciascuna sorgente di luce e bisogna anche tener conto di possibili effetti d'ombra applicando il test sulle intersezioni lungo le linee che vanno da un oggetto a ciascuna sorgente luminosa. Infine bisogna usare una qualche forma di riduzione, simile a un controllo automatico di esposizione, per assicurare che i valori dei pixel stiano entro un certo campo accettato dallo schermo.
Spero che questa parte dell'articolo abbia dimostrato che la matematica del ray tracing sia molto semplice e che usando un approccio vettoriale la geometria tridimensionale diventi molto più comprensibile.

Sfumatura omogenea

Si era accennato al fatto che esistono semplici tecniche che permettono l'eliminazione dell'aspetto sfaccettato di oggetti che sono formati da molti poligoni. Uno di questi metodi è chiamato sfumatura di Phong dal nome di uno dei pionieri della computer graphics.

Abbiamo visto come la luminosità di una superficie dipenda da molti fattori:

1) Dal colore e dalla luminosità della luce incidente.

2) Dalla distanza da ciascuna sorgente di luce.

3) Dal colore della superficie stessa.

4) Dall'angolo tra la luce incidente e la normale alla superficie.

Se due facce adiacenti sono dello stesso colore, ed entrambe sono distanti dalle sorgenti luminose, allora il quarto fattore ha il maggior peso nella valutazione della differenza di luminosità tra le due facce. La normale alla superficie varia bruscamente quando la visione dell'osservatore passa da una faccia all'altra, sebbene sembrino essere parallele.

Si può ottenere l'illusione di una curvatura omogenea se invece di usare il vettore normale alla superficie se ne usa uno artificiale. La Figura 3 mostra come farlo.
Si calcola una normale media in ciascun angolo partendo dai valori delle facce che lo circondano. Su ogni faccia la normale fittizia è calcolata interpolando tra i valori delle normali poste agli angoli della faccia stessa.

Chiunque abbia usato l'opzione di sfumatura di Phong in Sculpt 3D sarà rimasto colpito dal modo in cui oggetti costruiti abbastanza grezzamente sembrassero curvi in maniera omogenea. Si perde l'illusione solamente lungo il margine del profilo dove è possibile discriminare la singola faccia dalle altre.

Le economie del ray tracing

Deve esserci un inganno !
Escludendo l'interfaccia con l'utente e gli agganci al sistema operativo per visualizzare un'immagine, l'intero codice di un programma di ray tracing includendo gli effetti di sfumatura, ombre, riflessioni e rifrazioni, non è più lungo di poche centinaia di linee in C. L'inganno è la lunghezza del tempo richiesto per rappresentare un'immagine.

Un immagine sull'Amiga in modo HAM interlacciato ha più di 150.000 pixel. Tipiche scene di Sculpt 3D contengono spesso più di un migliaio di oggetti triangolari. Con un insieme di poche sorgenti luminose bisogna tenere conto di circa un miliardo di test sulle intersezioni tra oggetti e raggi.
Ciascun test consiste in alcune operazioni aritmetiche in virgola mobile e, facendo una stima del tempo impiegato per produrre una singola immagine, si scopre che ci vorrebbe più di un anno di calcoli. Fortunatamente esistono diverse vie per ottimizzare il processo di ray tracing.

La maggior parte dei test sulle intersezioni fallisce; infatti se una scena è composta da molti oggetti la maggior parte di essi sarà piccolo e quindi con una bassa probabilità di intersecare un raggio. Se il programma è in grado di valutare relazioni spaziali può scartare molte delle potenziali intersezioni valutandole impossibili.
Per esempio se degli oggetti molto vicini tra di loro sono considerati come un unico gruppo è possibile controllare l'intersezione di un raggio con il gruppo, visto come una singola entità, e se il raggio non lo interseca non c'è la necessità di verifica per ogni singolo componente. I gruppi possono essere organizzati gerarchicamente; molti raggi falliscono il test con il membro a più alto livello gerarchico, così gli oggetti a livello più basso non devono più essere controllati.

Se alcuni oggetti sono opachi bisogna solamente trovare quello che un dato raggio interseca per primo. Ordinando gli oggetti in base alla distanza è possibile avere dei notevoli incrementi di velocità di esecuzione.
Analogamente quando si fanno dei test per ottenere le ombre è sufficente sapere che un dato oggetto proietta un'ombra e non c'è bisogno di cercarne un altro.

Un altra tecnica molto efficace per l'ottimizzazione dei programmi di ray tracing è basata sul concetto di coerenza. Il concetto si basa sul fatto che se un oggetto contribuisce alla formazione dell'immagine in un certo pixel è probabile che contribuisca anche nel pixel a esso adiacente.
Una tecnica simile si può anche applicare alle ombre; è sempre conveniente controllare il prima possibile un oggetto che proietta un'ombra perché ci sono buone probabilità che la sua ombra influenzerà anche il prossimo pixel.

Per ulteriori informazioni

Sebbene si possano trovare molte immagini generate con la tecnica del ray tracing in libri e articoli di computer graphics, in molti casi alla teoria del ray tracing è dedicata solamente una breve descrizione. Ciò è dovuto essenzialmente al fatto che la teoria è una rapida transizione da ciò che è elementare a ciò che è misterioso.
Chi fosse interessato a conoscere gli ultimi sviluppi sull'argomento può consultare delle pubblicazioni dell'IEEE o dell'ACM.

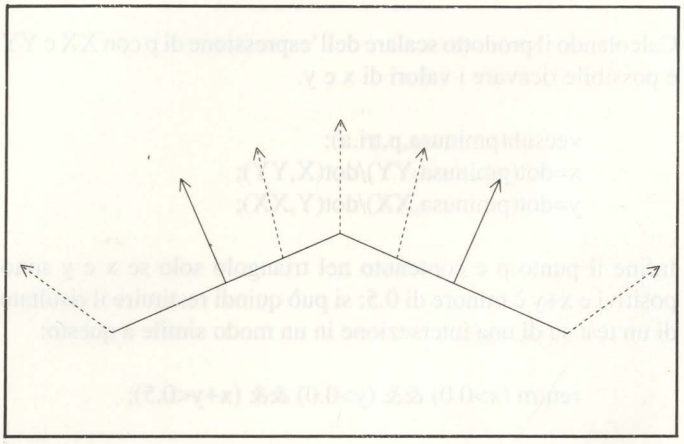


Figura 3: Tecnica di sfumatura di Phong. Le frecce continue rappresentano le normali reali mentre quelle tratteggiate rappresentano le normali fittizie usate per produrre l'illusione di linee arrotondate.

Con l'intento di distribuire immagini, software, articoli sul ray tracing e su argomenti legati a esso, sto pubblicando "Ray Tracing Newsletter".
Ciascun numero è distribuito su di un disco in formato Amiga e costa $6.00. Attualmente sono disponibili quattro numeri e due sono in preparazione.
Per ulteriori informazioni contattarmi direttamente al P.O. Box 579, Sandia Park, New Mexico 87047.

L'ultima visita all'Arp.Library

di Scott Ballantine e Charly Heath

Come creare altri processi ed altro ancora.

Scott Ballantyne è un programmatore attualmente interessato alla progettazione di sistemi operativi e di linguaggi per computer. Ha ricoperto un ruolo di primo piano negli sforzi di progettazione di ARP fino dall' inizio del progetto, scrivendo gran parte del codice di arp.library. Scott può essere raggiunto su CIS (70066,603) o BIX (sdb).

Charlie Heath, della MicroSmiths Inc., è l'autore di numerosi e popolari programmi per Amiga, in particolare TxEd, FastFonts e BlitzDisk. è anche l' iniziatore e coordinatore del progetto ARP. Charlie può essere raggiunto su CIS (74216,2117) o BIX (cheath).

Per coloro che fossero così sensibili da preoccuparsi per la fine di ARP, che potrebbe sembrare implicita dato il titolo di questo articolo, lasciateci assicurare che non è questo il caso. ARP è vivo e sta bene, grazie. I nostri obiettivi iniziali sono stati ampiamente raggiunti: è finalmente possibile per utenti e programmatori lavorare in un ambiente libero dal BCPL e i secondi hanno adesso a disposizione una consistente libreria di funzioni che può rendere la vita assai più facile.

Il titolo si riferisce al fatto che, a meno di imprevedibili sviluppi, arp.library è completa e non verranno aggiunte nuove funzioni. Con l'aggiunta delle funzioni nella versione 1.1 della libreria pensiamo che tutti gli aspetti principali siano stati coperti, almeno per quanto concerne il fornire le funzioni di cui in generale molti programmatori sentono la mancanza. Continueremo a correggere gli eventuali bug e ad apportare ritocchi e migliorie alla libreria ma non ci saranno sostanziali aggiunte fino a quando un numero molto più elevato di Amiga disporranno di memoria aggiuntiva.

L'evoluzione di ARP si sta muovendo verso nuovi concetti e nuovi programmi. Vogliamo che gli utenti abbiano la possibilità di fare cose che prima non erano possibili. Probabilmente ARP partorirà nuove shell, degli spooler per la stampa (buffer che si occupano di mandare in stampa i dati di un programma, lasciandolo libero di continuare a lavorare senza attendere la fine della stampa stessa) e dei programmi alternativi al Workbench. In questo momento, infatti, stiamo provando una versione preliminare del programma ARP che sostituirà il Workbench. è veramente impressionante e sembra molto promettente. Lo sviluppo di questo programma è nelle mani di Ken Salmon, unitosi agli sforzi del progetto ARP per la versione 1.1.

Il livello di accettazione di ARP

Il responso positivo di utenti e programmatori ad ARP è stato molto eccitante e piacevole. Una quantità di software commerciale e di pubblico dominio è già stato distribuito con arp.library e tutto lascia pensare che altro ne seguirà. I programmi che utilizzano arp.library sono particolarmente graditi perché tendono a essere più piccoli e più stabili. In genere quello che gli utenti apprezzano di più è il risparmio di spazio che si ottiene sul disco utilizzando i comandi di ARP. è molto comodo potere recuperare 50K sul Workbench o su un disco applicativo. Se è ovviamente impossibile fare funzionare un cattivo programma grazie ad ARP, è invece possibile rendere un programma, che funzioni già correttamente, più compatto e affidabile.

Possiamo vedere un buon esempio di ciò facendo un paragone fra il programma FaccII (ASDG) e BlitzDisk (MicroSmiths). Mentre entrambi i programmi sono eccellenti, BlitzDisk (che utilizza arp.library) è considerevolmente più piccolo, appena 2484 byte, mentre FaccII (che non utilizza arp.library) ha una dimensione di 9368 byte. Anche l'occupazione di memoria di questi due programmi durante il loro utilizzo è differente: BlitzDisk usa circa 2K, mentre FaccII ha bisogno di circa 19K, più RAM di quanta ne occupano BlitzDisk e arp.library messi insieme.

L'utilizzo di ARP migliora generalmente anche le prestazioni. Il Workbench standard di Amiga costringe i floppy drive a emettere cigolii e rumori di grattugia mentre va alla ricerca dei file .info e dei vari oggetti utilizzati dal file system. Il Workbench sostitutivo di ARP parte velocemente e silenziosamente, senza rumori eccessivi, principalmente perché utilizza le funzioni di ARP, funzioni che sono efficienti nell'esaminare dischi e liste di periferiche.

L'accettazione di ARP da parte della Commodore è materia di un negoziato ancora in corso, quindi non possiamo riferirne oltre. Possiamo comunque dire che siamo ancora interessati a permettere l'utilizzo di ARP da parte della CBM e che loro sembrano essere interessati a ottenerlo. Se siete utenti di ARP, potreste volere mandare una lettera a Max Toy o a Gail Wellington nella sede centrale di West Chester per segnalare il vostro gradimento.

Naturalmente, anche se ARP non dovesse finire su un futuro disco del Workbench, sembra che la maggior parte degli obiettivi di ARP siano stati raggiunti. Molto del software Commodore versione 1.3 (che, mentre scriviamo, deve essere ancora pubblicato

ufficialmente) contiene migliorie che sono state introdotte per la prima volta con ARP, anche se rimangono comunque ben lontane da quelle di ARP in alcune aree significative. Speriamo che la versione 1.4 contenga ARP o che questa realizzi gli obiettivi di ARP con maggiore completezza, mediante l'introduzione della gestione di processi nel senso prorio del termine e l'utilizzo di ragionevoli interfacce di linguaggio. In ogni caso, una stima di tipo conservativo del tempo che bisognerà aspettare per la release 1.4 del software di sistema ci porta lontano di uno, o più probabilmente, di due anni. Questo significa che un vero controllo dei processi, per non menzionare molti degli altri miglioramenti che ARP fornisce subito ad Amiga, sono ben distanti nel tempo.

In definitiva, ARP, con la sua larga accettazione fra gli utenti, così come fra i programmatori, rappresenta un messaggio per la Commodore, che esprime la necessità che le cose vengano migliorate. Il fatto che decidano di utilizzare le nostre soluzioni, o che decidano di rifare il nostro lavoro internamente, non è importante nel grande schema delle cose. ARP è iniziato come progetto per migliorare Amiga e, sia che rimanga uno sprone al miglioramento, o che diventi parte della release ufficiale del software, avrà comunque raggiunto il suo obiettivo più importante.

Cosa c'è di nuovo in ARP v1.1 ?

A parte le correzioni di alcuni bug, sono stati notevolmente migliorati GADS() (il parser per gli argomenti della linea di comando) e specialmente FileRequest(). GADS() accetta adesso un nuovo tipo di specifica della sintassi d'uso che permette di dichiarare un numero qualsiasi di argomenti, senza porre più limiti artificiali e senza quelle sgradevoli serie di virgole. FileRequest() è stato enormemente migliorato, tanto da meritare un posto in questo articolo, alla pari delle nuove funzioni di cui parleremo.
Arp.library adesso contiene anche alcune routine di conversione delle date che accettano i vari formati internazionali come input e come output, dando ad ARP e ad Amiga un tocco veramente internazionale.

A nostro parere, l'aggiunta più eccitante che sia stata fatta ad arp.library è la possibilità di lanciare un processo figlio (child process). Si può così iniziare l'esecuzione di un processo separato dal proprio programma, continuando poi a occuparsi dei fatti propri.
Non avremo bisogno di effettuare alcuna operazione di clean-up, in quanto il processo figlio si prenderà cura di sé stesso, fino a permettere al genitore (parent process) di terminare prima del figlio.
Questo non è mai stato possibile prima d'ora su Amiga e solo arp.library ci offre questa opportunità.

Abbiamo anche aggiunto il supporto per i programmi residenti e formulato una proposta per uno standard che riguardi questi programmi, in grado di funzionare sia con dati 'puri' sia con quelli 'non puri'. Altri hanno seguito approcci differenti per gestire i programmi residenti e in seguito parleremo dei pro e contro di ciascuno.

Controllo dei processi sotto AmigaDOS

Prima di mostrare la soluzione di ARP al problema del controllo dei processi su Amiga è opportuno chiarire ciò che la macchina offre di serie per risolvere questo problema. Come dovremmo tutti sapere, c'è una distinzione fra i processi dell'Exec e i processi del DOS. Si suppone che la parola 'task' venga utilizzata per riferirsi a un processo dell'Exec, mentre la parola 'processo' sia impiegata per i processi del DOS, ma in pratica spesso non succede e probabilmente non sarà così nemmeno qui. Faremo generalmente riferimento ai processi del DOS e verrà utilizzata l'espressione 'task dell'Exec' per riferirsi ai 'task'.

Bisogna anche fare un'infelice distinzione fra i processi del Workbench e del CLI. La differenza alla quale ci riferiamo non risiede nel tipo d'interfaccia utente, ma nella mancanza di uniformità e di integrazione fra l'ambiente Workbench e quello CLI dal punto di vista del programma. I canali di input/output fra i due ambienti sono problematici e inconsistenti, mentre le tecniche di controllo dello stack implementate per i processi CLI non funzionano assolutamente per quelli Workbench.

L'AmigaDOS mette a disposizione due funzioni per creare processi, CreateProc() ed Execute(), ognuna con degli specifici inconvenienti. CreateProc() può essere utilizzata per creare un processo asincrono e funziona abbastanza bene pur rimanendo entro i limiti della sua implementazione. Il problema principale di CreateProc() è rappresentato dal fatto che non viene eseguito nessun tipo di pulizia (clean-up) automatica una volta che il processo figlio sia terminato. Il programma genitore deve aspettare che il figlio abbia finito per potere rilasciarne la memoria occupata. Un problema direttamente legato al precedente consiste nel fatto che CreateProc() non si preoccupa neanche di avvertire quando il figlio ha terminato. Bisogna, quindi, far sì che il figlio invii un messaggio al genitore prima di uscire, oppure commettere il peggiore peccato contro il multitasking effettuando il poll di una lista di sistema per vedere se il figlio è ancora attivo. Questo limita l'utilizzo di CreateProc() ai programmi con i quali possiamo preventivamente prendere accordi speciali. Non è quindi possibile, tenendo anche conto che le dimensioni dello stack si trovano in un luogo differente da quello del CLI figlio, creare un'interfaccia sicura di input/output per i programmi.
CreateProc() è la funzione chiamata dal Workbench per lanciare i programmi ed è largamente responsabile della stupida e arbitraria distinzione fra programmi Workbench e programmi CLI che esiste su Amiga.

Per lanciare programmi CLI bisogna chiamare Execute(). Questa funzione creerà i necessari processi CLI, ma questi non saranno asincroni. Il nostro processo (il genitore) verrà messo a dormire fino a quando Execute() ritornerà con un messaggio di errore o fino a quando il processo figlio sarà terminato. Inoltre, è impossibile ottenere un valore di ritorno valido da Execute(); la triste verità è che non potremo mai essere sicuri se il programma che abbiamo tentato di lanciare sia stato effettivamente eseguito. Questo accade perché Execute(), in realtà, carica il programma Run dalla directory C ed è quest'ultimo che si occupa direttamen-

te di fare partire l'altro processo. Se Execute() non riesce a trovare il comando Run ci restituirà il valore 0, indicando il fallimento del suo compito, ma se Run non riuscirà a trovare il nostro programma, o se fallirà per una qualsiasi altra ragione, non ne verremo mai informati. Se proveremo poi a eseguire un processo CLI di background utilizzando un comando come "Execute("Run Type File to PRT:",0L,0L);" ci troveremo in un doppio guaio, dal momento che Run verrà chiamato in causa due volte. Si dice che la pazzia abbia un'origine molto semplice e deve essere vero, dal momento che questa semplice linea di codice mi ha spinto a occuparmi di ARP. Mi sono collegato a BIX per sfogare pubblicamente la mia ira nei confronti dell'AmigaDOS, ho scoperto il progetto di Charlie Heath e mi sono arruolato.

Quanto detto rappresenta solo un graffio sulla superficie dei problemi che si possono manifestare utilizzando Execute(). I due programmi che seguono illustrano le difficoltà causate dalla gestione inconsistente di input e output fra Workbench e CLI. Il primo programma scrive semplicemente un piccolo messaggio nel flusso dei dati verso l'esterno (output stream).

```
main()  /* Figlio.c */
{
  long Write(), OutPut();
  Write(Output(),"Hello World!\n",13L);
}
```

Il secondo programma carica e lancia il primo programma.

```
main()  /* Genitore.c */
{
  long Execute();
  Execute("RAM:Figlio",0L,0L)}
```

Compilate questi due programmi e create un'icona per il Genitore (per esempio copiando il file Clock.info come Genitore.info), quindi cliccate sull'icona Genitore per attivare il programma. A questo punto può succedere una delle tre cose:

1) Il computer va in crash.
2) Il computer rimane in attesa indefinita (come se non fosse successo niente).
3) Vedremo il messaggio "Hello World!" nel CLI che abbiamo utilizzato per lanciare il Workbench.

Se si è avverata la previsione numero 3, chiudiamo quel CLI utilizzando il comando EndCLI, quindi clicchiamo nuovamente sull'icona del Genitore un po' di volte. Nella maggior parte dei casi sembrerà che non stia succedendo nulla, ma proviamo adesso ad aprire un nuovo CLI e a dare il comando Status. Sorpresa! Vedremo che improvvisamente abbiamo molti CLI di background, tutti con la dicitura "No Command Loaded" ("Nessun comando in esecuzione"). L'unico modo di recuperare le risorse associate a ciascuno di questi CLI è di fare un reboot (CTRL-A-A).

Potremmo decidere di utilizzare gli altri argomenti della funzione Execute() per specificare i file handle per l'input e per l'output, ma questo genera a sua volta una serie di problemi. Uno di quelli più strani che si possono presentare è illustrato da questo programma:

```
main() /* Problema.c */
{
  long Execute(), Input(), OutPut();
  Execute("Type Problema.c",Input(),OutPut() );
}
```

Per vedere di quale problema si tratta, proviamo a dare un comando come questo:

```
1> Problema <Problema.c
```

Finiremo probabilmente per ricevere il misterioso messaggio:

```
Unknown command main()   (il comando main() è sconosciuto)
```

Questo succede perché Execute() insiste nell'interpretare un flusso di input non nullo come una serie di comandi! Naturalmente tutto questo richiede che ci sia il genitore ancora attivo a gestire i file handle per il figlio. Possiamo quindi abbandonare completamente l'idea di specificare i file handle di input e output per un processo indipendente.

C'è infine un problema straordinariamente irritante causato da Execute(). Non c'è modo di fare reagire un processo figlio a una richiesta di interruzione generata con CTRL-C. Infatti, nella maggior parte dei casi, il segnale CTRL-C verrà inoltrato al genitore. Utilizzando Execute() non c'è quindi modo di stabilire se l'utente che ha generato il segnale intendesse interrompere il processo genitore o il figlio!

Un ambiente integrato

Uno degli aspetti migliori delle funzioni di ARP per il controllo dei processi consiste nella loro completa integrazione con le altre parti interessate del sistema operativo. Per esempio, sia ASyncRun() che SyncRun() esaminano la lista dei programmi residenti mantenuta da ARP e successivamente il comando PATH associato al CLI durante la ricerca del programma da eseguire. Questo consente alle shell costruite con arp.library di funzionare correttamente con tutti i programmi dell'utente. Per esempio, anche il programma ARP che sostituirà il Workbench cerca nella lista dei programmi residenti e nelle directory specificate dal comando PATH, cosa che il Workbench attuale non fa. Per inciso, il Workbench di ARP si accorgerà anche di un eventuale comando Assign eseguito dal CLI.

ARP costruisce anche un frame nello stack che può essere utilizzato per il controllo dello stack stesso. Il frame utilizzato è quello documentato nell'AmigaDOS Developer's Manual (e utilizzato ben di rado dall'AmigaDOS). Il problema dell'input/output ha richiesto un approccio di tipo differente. Possiamo richiedere il caso di default che attiverà i normali handle di I/O per ciascun ambiente di lavoro, oppure quello alternativo che creerà

sempre un ambiente standard di input/output, eseguendo il processo nella propria finestra e permettendogli di rispondere a interruzioni come CTRL-C.

Non saremo comunque legati a nessuna di queste due alternative. Potremo infatti specificare uno o più file handle per il figlio, oppure addirittura NULL, se lo desideriamo. Infine, mentre il processo CLI asincrono è il tipo di processo creato di default dalle funzioni dell'AmigaDOS (ed è anche l'unico, attualmente, che si può creare), con ARP possiamo scegliere di creare anche un processo Workbench e persino di suddividere i pezzi del nostro programma in altrettanti processi!

ASyncRun()

La chiamata di funzione che viene utilizzata per fare tutto ciò è ASyncRun() e richiede tre argomenti: il nome del comando da eseguire, gli argomenti che si vogliono passare al figlio e un puntatore a una struttura chiamata ProcessControlBlock. ASyncRun() sostituirà spesso i propri valori di default agli argomenti contenenti NULL della struttura ProcessControlBlock. è quindi molto importante che tutte le variabili non utilizzate vengano messe a zero.

Prima di continuare dovremmo segnalare un bug che si è infiltrato nel file header arpbase.h della prima versione di ARP. In questo file ci sono due dichiarazioni della variabile pcb_SplatFile, mentre dovrebbe essercene una sola. La seconda va cancellata, lasciando quella dichiarata all'interno dell'unione pcb_Console. Le versioni più recenti (come quella distribuita da Transactor per Amiga) non presentano questo problema.

La maniera più semplice di utilizzare ASyncRun() è illustrata nel programma Defaults.c che appare alla fine di questo articolo nel listato 1. Dovrebbe essere abbastanza istruttivo il fatto di lanciare alcuni di questi esempi sia da CLI che da Workbench. Defaults.c si limita a stabilire le dimensioni dello stack del processo e la sua priorità (operazioni sempre necessarie) e si rimette al giudizio di ASyncRun() per il resto della struttura. Se il programma è stato lanciato da CLI verrà creato un processo CLI di background molto simile a quello creato dal comando Run dell'AmigaDOS. Il figlio condividerà la finestra di uscita con il genitore e utilizzerà il device NIL: come input. A questo punto l'unico modo di mandare un segnale al task è quello di utilizzare il comando Break. Nel caso in cui il programma è stato lanciato da Workbench, ASyncRun() metterà a uno il bit di controllo PRB_STDIO e aprirà una console window per il processo figlio.

Se vogliamo che il processo sia interattivo e che giri nella propria finestra, l'approccio più facile consiste nel mettere a uno il bit PRB_STDIO nella variabile pcb_Control.
Questo causa lo stesso tipo di comportamento sia da CLI sia da Workbench, provocando la creazione di un ambiente completo e standard per l'I/O e di una console window che passerà i segnali CTRL-C al processo giusto. Un esempio di come ottenere tutto questo è contenuto nel semplice programma stdio.c (vedere listato 2).

Specifichiamo i nostri handle di input/output

ASyncRun() ci permette anche di aprire i nostri file di input e/o output per il figlio e si preoccupa di chiuderli quando il figlio ha terminato. Se vogliamo ottenere questo comportamento, dovremo evitare di mettere a uno il bit PRB_STDIO, dal momento che questo costringerebbe ASyncRun() a ignorare i valori di pcb_Input e pcb_Output. Del resto, non siamo neanche costretti a fornire necessariamnete entrambi questi handle, in quanto ASyncRun() si prenderà cura al meglio delle proprie possibilità di quello che verrà omesso.
Un possibile problema che potrebbe verificarsi qualora prendessimo, nel modo precedente, un po' più di controllo sulle cose, si presenterebbe nel caso sfortunato che un figlio di tipo Workbench eseguisse un Open("*",MODE_mumble). Questo causerebbe la sospensione a tempo indefinito (hang) del processo figlio, in quanto la funzione Open() non restituirebbe più il controllo. ASyncRun() non può fare niente per risolvere questo problema da sola, ma può fare qualcosa con il nostro aiuto. Se abbiamo aperto una finestra CON: possiamo passarne l'handle ad ASyncRun() nel modo seguente:

```
struct ProcessControlBlock pcb;
pcb.pcb_Console.pcb_SplatFile = Console_FileHandle;
```

ASyncRun() sistemerà le cose in modo che se il figlio chiamasse Open("*") verrebbe utilizzato il file di console contenuto in pcb_SplatFile. Notiamo che questo file non verrà chiuso automaticamente da ARP quando il figlio avrà terminato. Se vogliamo che ARP lo chiuda automaticamente dobbiamo mettere a uno il bit di controllo PRB_CLOSESPLAT nella variabile pcb_Control e, ovviamente, in questo caso non dobbiamo chiudere da soli il file o assumere che sia disponibile per il nostro utilizzo. Tutte queste cautele sono necessarie solamente non utilizzando le regolazioni di default e solo nel caso di un figlio generato da un processo di tipo Workbench.

Ci sono infine dei casi in cui è desiderabile avere il completo controllo degli handle del figlio. In questi frangenti si può mettere a uno il bit PRB_SAVEIO nella variabile pcb_Control.
ASyncRun() non altererà le nostre specifiche e non chiuderà i file all'uscita.
Utilizzando questo bit si possono passare dei file handle contenenti NULL al figlio, cosa che si è dimostrata utile in alcune circostanze. Notiamo che questa opzione è indipendente dalla pcb_SplatFile e PRB_CLOSESPLAT di cui abbiamo parlato prima ed è pertanto possibile utilizzare contemporaneamente PRB_SAVEIO e specificare una console window di default.

Sincronizzazione di processi

Mentre ASyncRun() si occupa unicamente di eseguire processi indipendenti, in alcuni casi può essere utile sapere quando i processi sono terminati. Utilizzando ASyncRun() si possono generare alcuni task, fare delle elaborazioni per conto proprio e aspettare che i task finiscano il loro compito. Possiamo infatti

chiedere ad ASyncRun() di mandarci un messaggio quando il processo figlio è terminato. Il messaggio spedito, in questo caso, viene chiamato ZombieMsg e contiene svariati frammenti di informazione utile, come lo stato del processo al suo termine, lo stato del sistema al momento della terminazione e il momento esatto in cui è finita l'esecuzione. C'è anche lo spazio per collegare qualsivoglia nostra struttura dati. Il coordinamento di sotto-processi è illustrato in sync.c (vedere listato 3).

Come si può vedere in sync.c, abbiamo solamente bisogno di ottenere una MsgPort, inizializzare ReplyPort nella porzione riguardante l'ExecMessage di ZombieMsg con questa porta, lanciare il processo e attendere una risposta. Sebbene siano state utilizzate due strutture PCB, per convenienza e per chiarezza, non ce n'é realmente bisogno.
ASyncRun() non usa ProcessControlBlock dopo avercela restituita, quindi possiamo riutilizzarla come desideriamo. D'altra parte avremo ovviamente bisogno di tanti ZombieMsg quanti sono i processi, dal momento che questi sono posseduti dai figli fino a quando non ci vengono restituiti.

La parte di codice nella quale aspettiamo gli ZombieMsg merita un commento approfondito, così come la parte che si occupa della pulizia finale. Normalmente ASyncRun() ci restituisce il numero del processo CLI del figlio appena generato. Se tuttavia c'è qualche problema, il risultato di ASyncRun() sarà uno dei codici negativi di errore documentati nei file header di ARP. Quello che succede nel caso di un errore è importante: ARP eseguirà le proprie operazioni di clean-up come se il figlio fosse stato eseguito normalmente. Questo vuole dire che, a meno di non modificare appositamente questo comportamento, tutti i file aperti verranno chiusi, il codice deallocato e così via. Tuttavia, siccome in un caso come questo il processo è morto già al ritorno da ASyncRun(), non dovremo attendere uno ZombieMsg che infatti non verrà inviato. Tutto si riduce a queste semplici regole:

* Mai effettuare il clean-up di un task, a meno di avere espressamente segnalato ad ASyncRun() che vogliamo occuparcene noi.

* Attendere uno ZombieMsg, precedentemente richiesto, solo nel caso ASyncRun() restituisca un valore positivo.

La parte restante del programma dovrebbe essere abbastanza auto-esplicativa. Viene utilizzata una struttura software creata appositamente, collegandola allo ZombieMsg di ogni task per tenere sotto controllo il nome del comando, il momento della dipartita e il codice restituito da ASyncRun(). Questa struttura può essere estesa o modificata a piacimento, dal momento che ASyncRun() ignora completamente la variabile UserInfo. In questo programma abbiamo anche utilizzato le routine di conversione datestamp=>stringa (ne riparleremo più avanti) per convertire i datestamp in una forma umanamente leggibile.

Altri tipi di processi

ASyncRun() crea di default un processo CLI. Quando abbiamo bisogno di generare un processo nello stile di CreateProc(), possiamo farlo mettendo a uno il bit di controllo PRB_NOCLI. Naturalmente questo ci causerà alcune complicazioni extra, in quanto la maggior parte dei programmi là fuori si aspetteranno di ricevere da noi un messaggio di start-up del tipo utilizzato dal Workbench, oppure non funzioneranno correttamente nel nostro ambiente. D'altro canto ASyncRun() si preoccuperà comunque di svolgere la maggior parte delle operazioni di clean-up in nostra vece, compresa la deallocazione del programma e la chiusura dei file handle. Se dovessimo quindi avere bisogno di ricorrere a CreateProc(), la cosa migliore, anche in questo caso, dovrebbe consistere nel sostituirla con ASyncRun().

Normalmente ASyncRun() restituisce il numero del processo CLI del figlio, qualora le operazioni di lancio siano terminate con successo. Tuttavia, non esiste un numero di CLI associato con processi non-CLI, quindi ASyncRun() restituisce zero se è stato utilizzato PRB_NOCLI e se non sono avvenuti errori. Nella variabile pcb_WBProcess della struttura ProcessControlBlock, inoltre, ci viene restituito il puntatore alla MsgPort del processo figlio, che possiamo utilizzare per comunicare col processo non-CLI che abbiamo appena creato.

Come esempio di queste operazioni, wbproc.c (listato 4) carica ed esegue il classico programma Clock con PRB_NOCLI attivato. Notiamo che il task genitore deve preparare e spedire un messaggio di start-up come quello del Workbench e deve inoltre attenderne la risposta.

Sebbene non fossimo costretti, abbiamo scelto di caricare e deallocare il programma da soli, utilizzando le funzioni ARP LoadPrg() e UnLoadPrg(). Queste sono le funzioni chiamate da ASyncRun() per trovare il comando che gli viene fornito come primo argomento. Queste gestiscono correttamente la lista residente e quella dei path, inoltre restituiscono gli stessi valori della funzione LoadSeg() dell'AmigaDOS, cioè un BPTR a un seglist oppure NULL qualora sia avvenuto un errore.

Se abbiamo già il programma a portata di mano (già caricato in memoria), possiamo comunicare ad ASyncRun() la sua posizione mettendo il BPTR al suo seglist nella variabile pcb_LoadedCode della struttura ProcessControlBlock. Questo ci permette di eseguire più volte la stessa copia del programma contemporaneamente, qualora fosse necessario farlo. In questo caso dovremo però aspettare che il figlio abbia finito per deallocarne il codice.

Notiamo che, anche se l'esempio wbproc.c non lo mostra, possiamo fornire gli handle di input/output anche per i processi di tipo CreateProc(). Come al solito ARP li chiuderà al nostro posto durante l'uscita.

Strano e meraviglioso?

In definitiva, ASyncRun() ci permette di trasformare pezzi del nostro programma in altrettanti processi separati. Questa è una tecnica comunemente usata dai task di basso livello dell'Exec, ma non può essere implementata facilmente con i processi DOS a causa delle esigenze dell'AmigaDOS in merito a BPTR, seglist e tutto il resto.

Per raggiungere questo obiettivo, bisogna installare un CPTR (nota: NON UN BPTR!), che punti all'inizio del nostro codice, nella variabile pcb_LoadedCode e bisogna poi mettere a uno il bit PRB_CODE nella variabile di controllo. Questo è tutto quello che bisogna fare. In questo caso, naturalmente, non possiamo uscire prima del figlio; ricordiamoci che siamo stati caricati insieme. A parte quest'ultimo particolare, possiamo fare qualsiasi cosa possa fare un task dell'Exec e qualsiasi cosa possa fare un processo del DOS, inclusa l'apertura di file e librerie, la redirezione dell'output verso la stampante e tutto quanto rimane, senza penalizzazioni nella velocità.

Un esempio di tutto questo si trova nel programma magic.c (listato 5). Una rapida occhiata al listato ci mostra come sia estremamente semplice preparare e utilizzare un sotto-processo di questo tipo. Le uniche novità riguardano la preparazione dell'ambiente per il sotto-processo. Gli utenti dell'Aztec C che utilizzano il modello small devono chiamare la funzione geta4() quando il processo viene lanciato, per inizializzare il puntatore del compilatore alla sua zona dati. Gli utenti del Lattice C che utilizzano il modello small devono utilizzare lo switch -y, che in pratica ha la stessa funzione.

Un punto importante da ricordare quando si fanno questo genere di cose con ASyncRun() è che bisogna essere molto cauti nel chiamare qualsiasi funzione appartenente alle librerie del compilatore. Questo è il motivo per cui MyProcess() esce chiamando Exit() che appartiene all'AmigaDOS e non exit() che appartiene alla libreria del compilatore. Il problema è che i programmi C cominciano generalmente con del codice speciale generato dal compilatore per allocare i buffer per i file e così via. Quando entriamo in MyProcess() non passiamo attraverso questo codice speciale, quindi non ci troviamo nell'ambiente creato dal compilatore. A meno di non ricrearlo noi stessi, è sempre meglio limitarsi a chiamare le funzioni di sistema.

Programmi residenti, come e perché

Normalmente, su Amiga, quando lanciamo ripetutamente e contemporaneamente lo stesso programma, vengono create più copie del programma stesso in memoria. I programmi residenti, invece, hanno bisogno di un'unica copia in memoria del loro codice, anche quando vengono lanciati più volte contemporaneamente. Questo può portare a sostanziali risparmi di memoria e di tempo, in quanto non bisogna attendere che il programma venga caricato ogni volta. Ovviamente tutto questo è molto attraente, soprattutto per programmi come un text editor o come alcuni dei comandi più usati come Dir e List. Probabilmente a causa di questo acceso interesse è stato escogitato più di un approccio per rendere i programmi residenti su Amiga. Per valutare e capire questi metodi è necessario capire le questioni e i problemi coinvolti al momento di eseguire programmi residenti.

Il criterio che seguono i programmi residenti

Dal momento che più di un processo potrebbe trovarsi a utilizzare lo stesso frammento di codice nello stesso momento, è ovviamente importante che il frammento di codice non modifichi sé stesso. La maggior parte dei programmi per Amiga soddisa questa condizione, sebbene ci sia un numero sempre maggiore di programmi che fanno uso del cosiddetto 'SegList Splitting', che è l'equivalente binario del tagliarsi la testa, specialmente per quanto riguarda il poter diventare residente.

Lo stesso discorso vale per i dati. Dal momento che più di un processo potrebbe utilizzare gli stessi dati contemporaneamente, è importante che nessun dato di tipo static venga modificato. I dati da cambiare devono essere tenuti nei registri, in memoria allocata o nello stack. Molti programmi per Amiga non soddisfano questa condizione e, se utilizzati come programmi residenti, si comporteranno in maniera imprevedibile o andranno in crash. I programmi che soddisfano questa condizione, invece, vengono chiamati programmi 'con codice e dati puri' (pure code and data) o semplicemente 'puri' (pure). Non ce ne sono molti. I comandi originali in BCPL sono programmi puri, come del resto lo sono i comandi sostitutivi di ARP. Ci sono probabilmente altri programmi puri in circolazione, ma sicuramente non sono una grande quantità. In ogni caso, non utilizzate la purezza di un programma come criterio per stabilire la sua bontà; il modificare dati di tipo static è considerato perfettamente accettabile, anche dai sacerdoti del software.

Adesso possiamo illustrare i differenti metodi per gestire i programmi residenti, compreso quello utilizzato da ARP. Ogni approccio è riconducibile a uno di due gruppi. Il primo gruppo eseguirà solo programmi puri, mentre il secondo gruppo tenterà, in qualche maniera, di permettere anche a programmi non puri di diventare residenti per mezzo della capacità di copiare i dati. Al primo gruppo appartengono tutte le shell commerciali, il CLI residente originale distribuito esclusivamente agli sviluppatori con la versione 1.2 e il CLI che verrà fornito con la versione 1.3 del sistema operativo. Al gruppo due appartengono per il momento solo due programmi: quello di ARP e un altro programma liberamente copiabile e distribuibile chiamato REZ, scritto dal mago del software Jim Goodnow II, autore del compilatore Manx C per Amiga.

I programmi del primo gruppo sono di limitata utilità. Ci sono molti programmi, come i text editor, che dovrebbero poter diventare residenti, ma che non vengono accettati dal primo gruppo. Alcune delle shell commerciali disponibili per Amiga, addirittura, non controllano i programmi loro specificati per vedere se siano veramente puri o meno. Questo modo di operare può portare a crash istantanei e ciò non è affatto piacevole o d'aiuto.

I residenti di ARP e REZ

Ci sono svariate somiglianze fra i programmi residenti di ARP e REZ. Entrambi gestiscono i programmi puri ed entrambi saranno generalmente in grado di trattare anche programmi scritti con un linguaggio di alto livello, a patto che sia stato utilizzato il modello small durante la compilazione.
Ciascuno ha la capacità di copiare i dati dei programmi non puri e implementa una protezione sotto forma di checksum per scoprire il codice non puro ed eventuali danni causati da programmi che

sbagliano.
Questi programmi differiscono invece di gran lunga per quanto riguarda i dettagli e l'approccio utilizzato. Il metodo di Goodnow consiste nel modificare, tramite la funzione SetFunction(), il vettore delle chiamate di libreria LoadSeg() e UnLoadSeg(). Quando il programma che l'utente vuole rendere residente viene caricato, le informazioni di rilocazione vengono immagazzinate in una libreria simile a quella dell'Exec, chiamata REZlib.library. Il codice del programma viene inoltre modificato in modo da puntare a una routine contenuta in REZlib.library, simile alla geta4(). Questo è un approccio molto intelligente, veramente consigliabile.
Il suo vantaggio principale consiste nel permettere a molti programmi disponibili di diventare residenti senza bisogno di essere ricompilati, al contrario di quanto richiesto invece da ARP. Il suo maggiore svantaggio consiste nell'avere un overhead (il numero di istruzioni di 'introduzione e preparazione' che devono essere eseguite prima di passare il controllo al programma vero e proprio) molto più alto di quello di ARP.

L'approccio attualmente utilizzato da ARP per i programmi residenti consiste nel fornire uno speciale header (intestazione) Resident all'inizio di un programma. Questo header, a parte il fatto di rendere possibile l'identificazione del programma che può essere reso residente, contiene le informazioni sulla dimensione dello stack e sulle zone dati utilizzate.
Questo header viene riconosciuto dal caricatore e dal gestore di processi contenuti in arp.library, i quali cooperano per allocare i dati e lo stack come dalle indicazioni contenute nel Resident header. I dati allocati vengono passati al programma tramite il registro a4.
ARP non si rifiuterà di caricare i programmi sprovvisti di questo header ma, a meno che questi programmi non siano puri, non riusciremo a lanciarli come programmi residenti per più di una volta. Da notare che ARP esegue sempre il checksum sui programmi prima di lanciarli per prevenire i crash (questa è una opzione che può essere attivata dall'utente anche in REZ).

Qual'è allora il migliore? Questa è una domanda alla quale è impossibile rispondere e noi infatti li usiamo entrambi. REZ gestisce i programmi che sono stati scritti prima che nascesse ARP e che fosse formulato lo standard per i programmi residenti, mentre ARP permette drastici risparmi di memoria con i programmi che supportano questo standard.
ARP richiede un piccolo sforzo addizionale ai programmatori, per cooperare con il proprio standard, mentre REZ non richiede alcun cambiamento.
D'altra parte, esistono dei programmi che REZ non può rendere residenti, compreso lo stesso compilatore Manx e il relativo programma di utilità Make. Richiedere quindi al programmatore un piccolo lavoro aggiuntivo per rendere il programma residente potrebbe non essere una cattiva idea.

Rendere un programma residente

Dal momento che i programmatori devono fare uno sforzo speciale per rendere il loro programma residente con ARP, una domanda naturale è quanto sforzo sia esattamente richiesto e quanta attenzione sia necessaria.
Le risposte sono, rispettivamente, non molto e non molta. Io utilizzo una versione modificata dell'editor MicroEmacs per gran parte del mio lavoro su Amiga e, come test, ho deciso di convertire questo programma in modo da renderlo residente. Il tempo complessivo impiegato è stato inferiore a un'ora. Adesso risparmio 50K ogni volta che utilizzo l'editor, un bel risparmio, soprattutto con quel che costa la memoria di questi tempi.

A che cosa bisogna fare attenzione quando si scrive un programma che si vorrebbe poter rendere residente? Probabilmente la difficoltà maggiore risiede nella inizializzazione dei dati di tipo static, come mostrato di seguito:

```
struct Foo
{
        struct Foo *next;
        ULONG dummy;
};

struct Foo foo_array[] =
{
        {
                &foo_array[1],  /* questo è il problema */
                0,
        },      {
                &foo_array[2]
                1,
        },
};
```

Questa situazione si presenta molto spesso nei programmi, specialmente su Amiga, dal momento che viene fatto grande uso di strutture (come i Menu) che devono essere collegati insieme. Il problema, con il tipo di inizializzazione precedente, si manifesta solo se dobbiamo scrivere in queste strutture.
Se viceversa dobbiamo solo leggerle, non ci sono problemi. In ogni caso, se un programma residente modifica queste strutture durante l'esecuzione, solo gli originali vengono alterati, non la copia che ARP ne fa per essere utilizzata dal programma. Se dovesse succedere ciò, ARP se ne accorgerà e genererà un requester per informarci della difficoltà.

Esiste una soluzione abbastanza semplice per questo problema: basta inizializzare tutti i riferimenti fra un oggetto e l'altro utilizzando un semplice loop durante l'esecuzione. Il nostro programma in questo caso girerà correttamente.

Jim Goodnow ha scoperto qualcosa alla quale bisogna fare attenzione, in quanto causerà dei problemi sia ad ARP sia a REZ:

```
char *bra;

bra = "bxx";
strcpy(bra+1,branches[index]);
```

Il problema è che il compilatore memorizza le dichiarazioni di


GRUPPO EDITORIALE JACKSON NUMERO UNO NELLA BUSINESS-TO-BUSINESS COMMUNICATION
INFORMATICA OGGI
SETTIMANALE 60
Anche le reti locali nel mirino di Olivetti
Wyse Technology obiettivo Italia
GRUPPO EDITORIALE JACKSON
MECCANICA OGGI
SETTIMANALE 1
SETTIMA
ELETTRONICA · AUTOMAZIONE · S
La fabbrica elettronica au
INFORMATICA
BIT
ANNO 11 N.97 Settembre 1988
SPECIALE UNIX
VETRINA Tutto su AppleFest
Compaq a 25 MHz
12X, il 12 con Totus
elettronica OGGI
Speciale: SOFTW DI SIMU
STRUMENTAZIONE &MISURE OGGI
PC Floppy + PC MAGAZINE
La soft-rivista per gli utenti di Personal Computer IBM, Olivetti e compatibili
SideKik Plus
Screen Designer
Lotus: Sistema Esperto finanziario
Block per Paradox
Summa, gestionale per tutti
PC WORLD MAGAZINE
La rivista per gli utenti di Personal Computer IBM, Olivetti e compatibili
Amiga Transactor
Articles
Columns
Trasmissione Dati e Telecomunicazioni
Il mensile dei sistemi e servizi di comunicazione, trasmissione dati e telematica
SUPER COMMODORE 64 e 128
GIUGNO 1988 ANNO 5 - N.18
CON DISCO
ROOT RACE
OIL DEFENSE
SKETCH PAD PLUS
ML CLONER
BASICALLY MUSIC
OMPUTER GRAFICA
APPLICAZIONI
AMIGA MAGAZINE
CONTIENE:
CONTIENE DUE FLOPPY DISK CON PROGRAMMI
PC GAMES
IN QUESTO NUMERO:
CONCENTRATION • FREESTYLIN
TIC TAC TOE • ULTIME NOVITA'
CALCIO INDOOR
N. 29 Settembre '88
Anno 4
Compu Scuola
La rivista di informatica nella didattica per la scuola italiana
NUMERO SPECIALE
dedicato agli Atti di
SCUOLA 2000
olivetti PRODEST
USER
LA PRIMA E UNICA RIVISTA INDIPENDENTE PER GLI UTENTI PC 128-PC 128 S-PC 1
PC 1
Pcuno, pcuno, come te non c'è nessuno
Xtree
PC 128 S
Sistema musicale
Le risposte di Elisa
PC 128
Sistemi di numerazione
Quadrato cinese
IL L.M. DEL PC 128
N.30 SETTEMBRE '88
Realizzazioni pratiche
Tachimetro per bicicletta
Audiometro
Controller per impianti di riscaldamento
Voltmetro digitale per MSX
Accordatore d'antenna

ABBONAMENTO JACKSON = SERVIZIO COMPLETO

Da quest'anno l'abbonamento alle riviste Jackson offre una serie innegabile di vantaggi e servizi: anzitutto lo sconto eccezionale del 40% sul prezzo di copertina, pressochè doppio rispetto al passato, che Jackson ha voluto proporre ai lettori

per celebrare il decimo anno di attività. Inoltre, abbonarsi a Jackson garantisce l'accesso a una rete multinazionale di informazioni, grazie al recente accordo azionario con la VNU Business Press Group, maggiore editore tecnico internazionale del settore. Ma c'è di più: **la Jackson Gold Card**, per l'identificazione immediata del codice abbonamento, sarà recapitata gratuitamente agli abbonati e permetterà al titolare di usufruire di molteplici servizi gratuiti quali: sconto del 20% fino al 28/2/1989 e del 10% dopo tale data, sul prezzo di copertina di libri e software Jackson, per acquisti effettuati direttamente dall'editore, oltre a una serie di sconti per acquisti vari presso librerie, computershop e altri esercizi convenzionati in tutta Italia.

In più, il titolare di **Jackson Gold Card** potrà ottenere sconti sui corsi di formazione della Jackson S.A.T.A., la scuola Jackson di Alte Tecnologie Applicate, oltre all'abbonamento gratuito a 6 numeri di uno (a scelta) dei tre settimanali Jackson: "E.O. News Settimanale di Elettronica", "Informatica Oggi Settimanale" o il nuovissimo "Meccanica Oggi", annunciato per l'inizio del 1989.

Infine, l'abbonato ha diritto all'invio personalizzato e riservato dei cataloghi libri e della nuova rivista **"Jackson Preview Magazine"**, con l'annuncio di tutte le novità editoriali Jackson.

GRUPPO EDITORIALE JACKSON

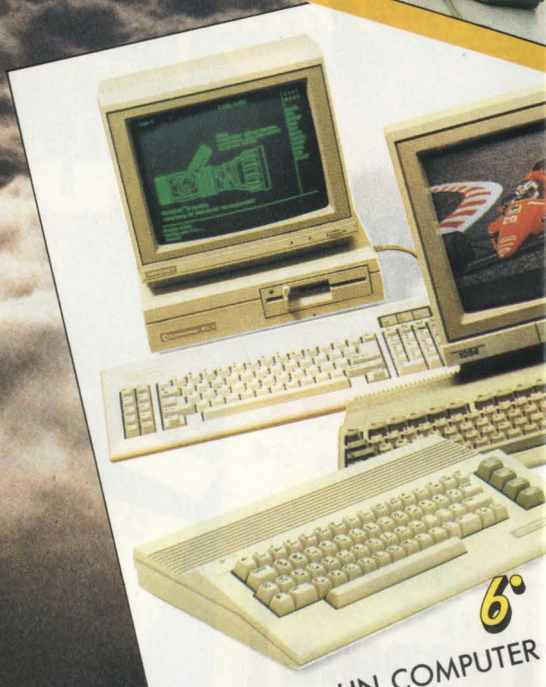
PRIMO NELLA BUSINESS-TO-BUSINESS COMMUNICATION

1° PREMIO

HONG KONG • BANGKOK • SINGAPORE

2° PREMIO

UN COMPUTER AMIGA 2000



6°

UN COMPUTER

UN FANTASTICO VIAGGIO
IN ESTREMO ORIENTE
IN COLLABORAZIONE CON:

ACENTRO TURISMO

swissair

Sheraton

I LEADER PER UN VIAGGIO DI SUCCESSO

dal 7° al 10° PREMIO

4 PACCHETTI SOFTWARE
"Commodore Software by CTO"

ABBONAMENTO JACKSON = FORTUNA STREPITOSA

AUT. MIN. RICH.

Abbonarsi alle riviste Jackson significa leggere il meglio, risparmiando il 40%, in informatica, elettronica e nuove tecnologie, ma soprattutto partecipare al grande concorso Jackson riservato agli abbonati, con la possibilità di vincere premi favolosi.

REGOLAMENTO DEL CONCORSO

1 – Il Gruppo Editoriale Jackson S.p.A. promuove un concorso a premi in occasione della Campagna Abbonamenti 1988/1989.

2 – Per partecipare è sufficiente sottoscrivere, entro il 31.3.1989, un abbonamento a una delle 30 riviste Jackson.

3 – Sono previsti 10 favolosi premi da sorteggiare fra tutti gli abbonati.

4 – Primo premio: un viaggio per due persone in Estremo Oriente, che prevede: passaggi aerei Swissair, pernottamenti in Hong Kong, Bangkok e Singapore, presso gli hotel: Royal Orchid Sheraton e Sheraton Towers della catena Sheraton Hotel, nonchè escursioni in luogo nelle tre suddette località.
Gli altri nove premi consistono rispettivamente (in ordine di esposizione) in:
1 computer Amiga 2000 completo di unità centrale con 1 MB di memoria, dischetto da 3" 1/2, tastiera, mouse, sistema operativo e monitor a colori 1084.
1 personal computer PC 20 III SERIE completo di unità centrale con 640 KB di memoria, dischetto da 5" 1/4, hard disk da 20 MB, mouse 1352, sistema operativo MS-DOS 3.20 monitor monocromatico e tastiera.
1 personal computer PC1 completo di unità centrale con memoria 512 KB, dischetto da 5" 1/4, tastiera, monitor monocromatico, sistema operativo MS-DOS 3.20 e GW-Basic.
1 computer Amiga 500 con 512 KB Ram e 256 KB Rom di memoria, sistema operativo e monitor a colori 1084.
1 computer "nuovo C64" completo di manuali e sistema operativo.
Dal settimo al decimo premio incluso, n. 4 pacchetti "Commodore Software by CTO".

5 – Gli abbonati a più di una rivista avranno diritto, per l'estrazione, all'inserimento del proprio nominativo tante volte quante sono le testate sottoscritte.

6 – L'estrazione dei 10 premi in palio avverrà presso la sede del Gruppo Editoriale Jackson entro il 30.5.1989.

7 – L'elenco dei vincitori, ad estrazione avvenuta, pubblicato su almeno 10 delle riviste Jackson. La vincita inoltre sarà pubblicata con lettera raccomandata a ciascuno dei sorteggiati.

8 – I premi verranno messi a disposizione degli aventi diritto entro 30 giorni dalla data dell'estrazione, ad esclusione del primo premio, il quale dovrà essere effettuato, compatibilmente con la disponibilità dei posti, in un periodo da definirsi, entro il 31.12.1989.

9 – Le spese di vitto relative al viaggio, nonchè l'eventuale controllo di manutenzione extra garanzia per i personal computer Commodore, saranno a carico dei rispettivi vincitori.

10 – I dipendenti, i familiari, i collaboratori del Gruppo Editoriale Jackson sono esclusi dal concorso.

Sempre quest'anno, il concorso abbonamenti Jackson prevede un primo premio veramente eccezionale: la possibilità di esplorare il misterioso Estremo Oriente, in un viaggio che unisce il fascino di una tradizione millenaria ad uno sviluppo tecnologico senza precedenti.

Il viaggio, di oltre dieci giorni per due persone, è studiato nei minimi dettagli, per offrire il miglior comfort e le migliori ospitalità ed è garantito da tre leader di primissimo livello: Acentro Turismo di Milano, Swissair e Sheraton Hotels.

Non solo. Ad altri nove abbonati fortunati, il Gruppo Editoriale Jackson, in collaborazione con Commodore Computer e CTO, riserva altri premi eccezionali, dalla più completa gamma di computer di successo: un favoloso personal computer Amiga 2000, un Commodore PC20 III serie, un Commodore PC1, un Amiga 500 e un nuovo C64, in palio dal secondo al sesto estratto. Quattro pacchetti "Commodore Software by CTO" saranno inoltre sorteggiati dal settimo al decimo premio.

Partecipare al concorso è semplice: basta abbonarsi a una o più tra le riviste Jackson (chi si abbona a più riviste ha, naturalmente, più possibilità di vincita), utilizzando la speciale Cartolina/Questionario, già predisposta e affrancata, da compilare in ogni sua parte e restituire all'editore.
Affrettatevi! Abbonatevi per vincere!

Commodore
LEADER IN PERSONAL COMPUTER

GRUPPO EDITORIALE JACKSON
PRIMO NELLA BUSINESS-TO-BUSINESS COMMUNICATION

ABBONAMENTO JACKSON = RISPARMIO ECCEZIONALE

Abbonarsi è semplice: basta compilare in ogni sua voce la speciale Cartolina/Questionario già predisposta e affrancata e rispedirla all'editore.

Per il versamento dell'importo dell'abbonamento, utilizzate, preferibilmente l'apposito modulo di C.C.P. già predisposto e allegato alla rivista.

SERVIZIO QUALIFICAZIONE LETTORI

SPECIALE: PER CHI ACQUISTA LE RIVISTE JACKSON IN EDICOLA

Da quest'anno il Gruppo Editoriale Jackson ha predisposto uno **Speciale Servizio di Qualificazione Lettori e Abbonati,** che prevede l'assegnazione di una serie di dati relativi agli interessi specifici di ognuno, per poter offrire un servizio adeguato alle reali esperienze di aggiornamento del lettore.

Tutti i lettori interessati allo **Speciale Servizio Qualificazione Lettori,** e quindi anche i non abbonati, devono restituire, compilata nella parte **Qualificazione lettori,** la **Cartolina Questionario** già predisposta e affrancata.

Per chi la spedirà, il Gruppo Editoriale Jackson garantisce fin d'ora **GRATUITAMENTE:**

- Jackson Silver Card, che offre tutti i vantaggi della Gold Card, esclusi gli sconti sui libri riservati agli abbonati.

- Invio gratuito del **Catalogo Generale Libri Jackson.**
- Invio gratuito della **Jackson Preview Magazine.**
- **Abbonamento gratuito a sei numeri,** a scelta tra le seguenti riviste settimanali: E.O. News Settimanale - Informatica Oggi Settimanale - Meccanica Oggi (pubblicato da febbraio '89)

Area	Testate	Numeri Anno	Tariffa abbonam.	Tariffa intera
Elettronica e automazione	EO News Settimanale	40 + 6 omaggio	£. 59.500	£. 100.000
	Elettronica Oggi	20	£. 60.500	£. 100.000
	Automazione Oggi	20	£. 60.000	£. 100.000
	Meccanica Oggi	40 + 6 omaggio	£.59.000	£. 100.000
	Strumentazione e Misure Oggi	11	£ 39.000	£. 66.000
Informatica e Personal Computer	Informatica Oggi Settimanale	40 + 6 omaggio	£. 61.000	£. 100.000
	Informatica Oggi mese	11	£. 33.500	£. 55.000
	BIT (quindicinale da Gennaio)	20	£. 48.000	£. 80.000
	PC Magazine	11	£. 32.500	£. 55.000
	PC Floppy	11	£. 79.500	£. 132.000
	Computergrafica e applicazioni	11	£. 39.500	£. 66.000
	Trasmissione dati e Telec.	11	£. 34.000	£. 55.000
	Compuscuola	10	£. 24.500	£. 40.000
Tecnologie e mercati	WATT (quindicinale da Gennaio)	20	£. 36.500	£. 60.000
	LAB. NEWS	10	£. 30.000	£. 50.000
	Industria Oggi	11	£. 34.500	£. 55.000
	Media Production	11	£. 46.500	£. 77.000
	Strumenti musicali	11	£. 32.000	£. 55.000
Hobby e Home Computer	Fare Elettronica	12	£. 36.000	£. 60.000
	Amiga Magazine disk	11	£. 92.500	£. 154.000
	Amiga Transactor	6	£. 25.500	£. 42.000
	Commodore Professional 64/128 disk	11	£. 85.000	£. 143.000
	Commodore Professional 64/128 cass.	11	£. 59.500	£. 99.000
	Supercommodore 64/128 disk	11	£. 79.000	£. 132.000
	Supercommodore 64/128 cassetta	11	£. 49.500	£. 82.500
	Olivetti Prodest User	6	£. 18.000	£. 30.000
	PC Software	11	£. 66.000	£. 110.000
	PC Games 5 1/4"	11	£. 93.000	£. 154.000
	PC Games 3 ½	11	£. 99.500	£. 165.000
	3 ½ Software	11	£. 99.000	£. 165.000

Lo sconto del 40% è stato calcolato, in certi casi, arrotondando le cifre in modo da differenziare le tariffe di ciascuna rivista per esigenze di gestione.

GRUPPO EDITORIALE JACKSON

PRIMO NELLA BUSINESS-TO-BUSINESS COMMUNICATION

stringhe effettuate in questo modo nel segmento che contiene il codice del programma, che quindi viene modificato dalla chiamata alla funzione strcpy(). La soluzione è semplice:

```
char bra[] = "bxx";
```

L'overhead determinante

Adesso che abbiamo visto quanto sia semplice fare in modo che i programmi possano essere trattati come residenti, qual'è la differenza finale fra ARP e REZ? Considerando l'editor MicroEmacs che abbiamo visto in precedenza, abbiamo un programma che occupa circa 32K, inclusi entrambi i dati e il codice. REZ deve salvare una certa quantità di informazioni extra riguardanti la rilocazione per ogni programma, cosa che ARP non deve fare, ma il vero risparmio risiede nell'overhead non visibile generato durante la allocazione di memoria per lo stack. Gli utenti generalmente fissano lo stack di sistema a circa 20K, il che vuole dire che ogni programma lanciato utilizzerà questa quantità di memoria. Ovviamente questo è molto più di quanto sia spesso necessario al programma che è stato lanciato. Il programma MicroEmacs utilizza uno stack di soli 4K, quindi il metodo di ARP per i programmi residenti in questo caso ci fa risparmiare 16K rispetto a REZ. Questo si somma a una serie di risparmi di memoria.

Qual'è allora il migliore? Come già detto prima, noi li usiamo entrambi. I nuovi programmi dovrebbero utilizzare l'header Resident di ARP a causa del suo overhead notevolmente ridotto. Queste due tecniche, comunque, lavorano bene insieme, quindi pensiamo che si debba integrare la compatibilità con REZ nello standard ARP. Potremmo così salvare le informazioni per la rilocazione dei programmi che non abbiano il Resident header, continuando a usufruire del risparmio di memoria con i programmi più recenti.

ARP diventa internazionale

In questa versione di ARP sono diponibili due routine per la conversione delle date. La prima, StampToStr(), prende una struttura DateStamp e la converte in una stringa che possa essere letta facilmente, mentre la seconda, StrToStamp() effettua l'operazione inversa. Entrambe queste routine possono convertire da e per uno qualsiasi di quattro formati per le date: il formato internazionale, quello americano, quello canadese (ed europeo) e quello standard dell'AmigaDOS. Per selezionare uno di questi formati abbiamo creato una variabile di ambiente (environment variable) chiamata 'dateformat'. Se questa variabile non è definita, viene utilizzato il formato AmigaDOS, altrimenti si decide in base a questa tabella:

dateformat	Formato	
0	AmigaDOS,	gg-mmm-aa
1	Internazionale,	aa-mm-gg
2	USA,	mm-gg-aa
3	Canada (Europa),	gg-mm-aa

Un valore illegale nella variabile dateformat causerà l'adozione del formato standard AmigaDOS. Se si fà precedere uno qualsiasi dei valori sopraelencati con il segno meno '-', verrà soppressa la visualizzazione di nomi come 'Monday' o 'Tuesday' ecc. nell'output di comandi come List. Tutti i comandi di ARP che hanno a che fare con le date (per esempio List, Date, SetDate) utilizzano la variabile dateformat, con i suoi valori, per controllare la forma del proprio input e output. Le funzioni di conversione delle date contenute in ARP non leggono il contenuto della variabile dateformat da sole, in quanto questo potrebbe causare una conversione non desiderata espressamente. Così siamo noi che dobbiamo preoccuparci di leggere il contenuto di dateformat per presentare all'utente un'interfaccia consistente.

La buona notizia sta nel fatto che è molto facile svolgere questa operazione. L'approccio canonico viene mostrato nell'esempio DoDate.c (listato 6). Bisogna notare che si deve controllare esplicitamente la stringa dateformat per rilevarvi l'eventuale presenza del segno meno, perché non possiamo affidarci al risultato eventualmente negativo che ci viene restituito da Atol(), dal momento che l'utente potrebbe scrivere:

```
set dateformat=-0
```

per richiedere il formato AmigaDOS senza sostituzioni. In questo caso noi perderemmo infatti il segno meno, in quanto il valore zero viene sempre restituito come numero positivo dalla funzione Atol(). L'unico altro punto degno di menzione è che i due flag DTB_SUBST e DTB_FUTURE hanno influenza solo su StampToStr() e StrToStamp() rispettivamente. Questo significa che non è necessario, per esempio, mettere a zero DTB_FUTURE prima di chiamare StampToStr().

Il FileRequester di ARP, parte II

Le cose semplici, funzionali e pulite sono le più apprezzate. Questa è la filosofia che sta dietro al file requester di ARP. Il requester nella sua versione base fornisce alle applicazioni un mezzo consistente e funzionale per ottenere un input dall'utente. Con la versione 1.1 di ARP è diventato possibile personalizzare il requester per particolari esigenze dei propri programmi.

Il funzionamento della maggior parte delle funzioni del file requester risultano ovvie all'utente, già dopo averle viste solo un paio di volte, ma ce ne sono anche un paio che non risultano così semplici, a meno che non si sia più informati sul loro conto. La prima consiste nella possibilità di visualizzare una lista delle periferiche disponibili (come df0:, df1:, dh0: e ram:). Normalmente il file requester visualizzerà solamente i file contenuti nella directory selezionata, ma cliccando il tasto destro del mouse il display conterrà anche i nomi delle periferiche fisiche e delle unità logiche create con il comando Assign. La maggior parte degli altri file requester utilizzano delle icone per ognuno di questi device, ma questo spesso finisce per rendere il display ingombrante o incompleto.

La seconda caratteristica meno ovvia consiste nel poter limitare i file visualizzati a quelli che soddisfano un certo pattern. Se inseriamo un pattern alla fine del nome della directory, verranno

visualizzati solo quei file il cui nome soddisfa quel pattern. Questo può essere particolarmente utile quando si deve trovare un determinato file in una directory che ne contiene un numero molto alto. La valutazione del pattern viene fatta chiamando la funzione ARP PatternMatch, che accetta tutte le wildcard standard dell'AmigaDOS, oltre all'asterisco '*'.

L'utilizzo del file requester da parte dei programmatori è semplice come la definizione di una struttura dati per l'input e come la chiamata a una funzione. Ci sono alcune altre potenzialità che possono essere sfruttate, permettendo la personalizzazione del file requester per adattarsi alle esigenze del proprio programma, ma prima diamo un'occhiata alla versione base. Le seguenti linee di codice sono tutto quello di cui abbiamo bisogno per l'utilizzo del file requester nella sua versione più semplice:

```
/* definiamo queste strutture dati all'inizio del nostro
                                                programma */
char  FileName[FCHARS+1];
char  DirName[DSIZE+1];
struct FileRequest MyFileRequest =
{
  "Salve, sono il FileRequester!",          /* stringa di saluto */
  FileName,       /* puntatore al buffer per il nome del file */
  DirName, /* puntatore al buffer per il nome della directory */
  0L,      /* puntatore alla finestra in cui deve apparire
                                                o NULL */
  0,0,          /* flag per un controllo particolare (nessuno) */
  0L,           /* funzione speciale (nessuna in questo caso) */
  0L,                                        /* riservato */
};

...

/* all'interno del programma chiamiamo il file requester
                                          quando ci serve */
if (FileRequest(&MyFileRequest)==0L)
{
  Printf("Cancel!\n");
} else
{
  Printf("filename: %s, directory: %s\n",FileName,DirName);
}
```

In questo esempio i due buffer 'FileName' e 'DirName' riceveranno i nomi del file e della directory che l'utente ha selezionato nel file requester. Se l'utente seleziona CANCEL, la funzione FileRequest() restituirà zero. Se l'utente, invece, clicca su OK o preme RETURN nel gadget del nome del file, il valore restituito sarà il puntatore al buffer FileName. Notiamo che, anche quando viene selezionato CANCEL, il contenuto dei buffer FileName e DirName potrebbe comunque essere stato alterato dall'utente.

La prima aggiunta che potremmo volere fare nel nostro programma consiste nell'utilizzare la variabile fr_Window per fare apparire il file requester in una nostra particolare finestra di Intuition. Questo risulta utile per due motivi. Primo, se la nostra finestra si trova in un CUSTOMSCREEN il file requester apparirà su quello screen se la variabile fr_Window è stata inizializzata. Se la variabile non è stata inizializzata il requester apparirà sullo schermo del Workbench, cosa che può essere causa di confusione per il vostro utente che sta ancora fissando il vostro schermo custom e sta accarezzando l'idea di telefonare al servizio tecnico. Inoltre, inizializzando la variabile fr_Window, il file requester utilizzerà la stessa porta IDCMP della nostra finestra (a meno di non mettere a uno il bit FRF_NewIDCMP nel campo fr_Flags). Questo semplificherà la gestione degli eventi ed è comunque necessario se vogliamo scrivere una funzione speciale per gestire altri eventi generati all'interno della nostra finestra mentre utilizziamo il file requester (mettendo a uno il bit FRF_DoMsgFunc).

Un'altra semplice modifica che possiamo fare consiste nell'utilizzare il bit FRF_DoColor sempre nella variabile fr_Flags; questo ci consentirà di scegliere il secondo set di colori da utilizzare per il file requester. Per convenzione, il secondo set di colori viene normalmente utilizzato quando è in corso il salvataggio di un file, mentre i colori normali sono utilizzati durante il caricamento. In questo modo l'utente può rendersi conto immediatamente del tipo di operazione che è stata selezionata.

Possiamo anche utilizzare alcune delle possibilità speciali per la personalizzazione del file requester, mettendo a uno i bit corrispondenti alle nostre scelte nella variabile fr_Flags e inizializzando la variabile fr_Function. Quest'ultima è il puntatore alla funzione che verrà chiamata se si verificheranno una o più delle condizioni selezionate con i bit di fr_Flags.

La funzione fornita dall'utente verrà chiamata tutte le volte che si verificherà almeno una delle condizioni richieste. Quando la funzione viene chiamata, possiamo risalire alla particolare condizione che ha causato l'attivazione controllando il valore di 'bits', come mostrato in questo esempio:

```
My_fr_Function(bits,funcvalue)
long bits;
long funcvalue;
{
  switch(bits)
   {
    case FRB_NewWindFunc:
         faiqualchecosa();
         break;

    case FRB_AddGadFunc:
         faialtrecose();
         break;
     ...
    }
}
```

Le possibilità di personalizzazione comprendono:

FRF_NewWindFunc permette di alterare la struttura NewWindow che verrà utilizzata per aprire la finestra del file requester.

Potremmo volere cambiare la sua posizione o le dimensioni, per esempio, con questa funzione. I cambiamenti sono a discrezione del programmatore, ma attenzione a non fare cose strane, o il malocchio potrebbe colpire il nostro programma. In questo caso la variabile funcvalue è un puntatore alla struttura dati NewWindow.

FRF_AddGadFunc permette di aggiungere eventuali gadget personalizzati alla finestra del file requester. Questa funzione viene chiamata dopo che la finestra è stata aperta, ma prima che vengano visualizzati i gadget standard. Ogni gadget che aggiungiamo deve essere pronto a essere visualizzato e posizionato dove lo vogliamo fare apparire (a questo punto dovremmo probabilmente avere già ingrandito la finestra grazie alla chiamata generata da FRF_NewWindFunc).
In questo caso la variabile funcvalue è un puntatore alla Window del file requester. Notiamo che tutti i nostri gadget dovranno avere un valore della variabile GadgetID minore di 0x7680 o maggiore di 0x7700 circa.

FRF_GEventFunc permette di processare gli eventi speciali legati ai gadget che abbiamo aggiunto utilizzando FRF_AddGadFunc. In questo caso funcvalue contiene la variabile GadgetID del gadget selezionato.

Se abbiamo utilizzato FRF_DoWildFunc possiamo specificare una nostra funzione speciale per fare da 'filtro' per i nomi dei file, utilizzando qualsiasi wildcard ci faccia piacere. In questo caso la variabile funcvalue è un puntatore a un FileInfoBlock che contiene il nome del file richiesto; dovremo restituire il valore TRUE se vogliamo visualizzare questo file, oppure FALSE se non intendiamo farlo apparire. In realtà la faccenda DOVREBBE funzionare così, ma nella versione attuale di arp.library esiste un bug che costringe il file requester a ignorare il valore restituito dalla nostra funzione e da quella chiamata da FRF_GetEventFunc. Verranno pubblicate al più presto le correzioni per eliminare questo inconveniente.

FRF_DoWindFunc permette di processare per conto nostro i messagi IDCMP che vengono ricevuti per la finestra che ospita il file requester o per qualsiasi altra finestra stia condividendo la stessa porta IDCMP. Ogni volta che un IntuiMessage arriva nella porta IDCMP condivisa e questo messaggio non è indirizzato al file requester, viene chiamata la funzione fr_Function, con funcvalue che punta all'IntuiMessage ricevuto. Notiamo che il nostro programma deve rispondere a questo IntuiMessage se abbiamo messo a uno FRF_DoWindFunc!

Gran parte di queste cose sono illustate nel programma Freq.c (listato 7), il quale, dapprima mostra l'uso più semplice del file requester, poi lo attiva in uno schermo custom a bassa risoluzione. Questo esempio mostra anche come aggiungere un gadget e come fare alcune altre cose che potrebbero risultare interessanti e utili. Alcune di queste cose saranno pienamente funzionali solo quando verranno pubblicate le correzioni per eliminare i bug attuali.
Questo è tutto per quanto riguarda le capacità speciali. Se avete qualche estensione che vi piacerebbe vedere implementata nella prossima versione, mandateci una nota all'ARP support e noi la prenderemo in considerazione. Se volete utilizzare le funzioni speciali per la personalizzazione, farete meglio a consultare la ARP Programmers' Documentation, che contiene una descrizione tecnica più completa di tutti i casi speciali.

Questo è tutto ragazzi!

Questo è quanto, per ora. Tutte le persone che hanno lavorato per ARP dsiderano rigraziarvi per il sostanziale supporto, per le segnalazioni di bug e i suggerimenti che abbiamo ricevuto. ARP, più di altri progetti software, è cresciuto e ha coinvolto buona parte della comunità degli utenti di Amiga e noi speriamo che continui così anche per il futuro. Continuate quindi a mandarci i vostri suggerimenti e se avete scritto un programma con i fiocchi, con cui vorreste contribuire al progetto, perché non mettersi in contatto con noi? Il nostro indirizzo è:

ARP support c/o Microsmiths, Inc.
PO Box 561 Cambridge, MA 02140 USA

Esempi di programmazione ARP

Listato 1

```
/* defaults.c — Copyright (c) 1988 di Scott Ballantyne
*                    Usate e abusate quanto volete
*
* Usa TUTTI i valori di dafault per ASyncRun()
*
* Compilate normalmente, poi
* MANX:
* ln defaults.o -larp -lc
* LATTICE:
* blink arpc.o defaults.o to defaults lib lib:arp.lib lib:c.lib
lib:amiga.lib
*/


#include <exec/types.h>
#include <libraries/arpbase.h>
#include <arpfunctions.h>

struct ProcessControlBlock PCB =
{
  20000,          /* dimensione dello stack */
  0,              /* priorita' */
/* Il resto viene inizializzato a ZERO dal compilatore C,
quello che vogliamo */
};

VOID
        main()
{
        LONG rc;

        if (rc=ASyncRun("Type", "Defaults.C", &PCB)) <
0 )             Printf("ERRORE = %ld\n", rc);
        else
                Printf("Numero del processo =
                %ld\n", rc);
}
```

Listato 2

```
/* stdio.c — Copyright (c) 1988 di Scott Ballantyne
*                    Usate e abusate quanto volete
*
* Come impostare un ambiente di I/O CLI standard per
tutte le situazioni,
* sia per un processo Workbench che per uno CLI.
*
* Compilate normalmente, poi:
* MANX:
* ln stdio.o -larp -lc
* LATTICE:
* blink arpc.o stdio.o to stdio lib lib:arp.lib lib:c.lib
lib:amiga.lib
*/

#include <exec/types.h>
#include <libraries/arpbase.h>
#include <arpfunctions.h>

struct ProcessControlBlock PCB =
{
  20000,          /* dimensione dello stack */
  0,              /* priorita' */
/* Il resto viene inizializzato a ZERO dal compilatore C,
                                    quello che vogliamo */
};

VOID
        main()
{
        LONG rc;
/* Imposta STDIO usando la console window di vostra
                                                scelta */

  PCB.pcb_Control |=PRF_STDIO;
  PCB.pcb_Console.pcb_ConName = "CON:0/0/
                    320/200/STDIO Window!";

  if ( (rc = ASyncRun("Type", "?", &PCB)) < 0 )
                Printf("ERRORE = %ld\n", rc);
  else
                Printf("Numero del processo = %ld\n",
rc);
}
```

Listato 3

```
/* sync.c — Copyright (c) 1988 di Scott Ballantyne
*                  Usate e abusate quanto volete
*
*       Illustra:
*               Sincronizzazione tra processi usando gli
                ZombieMsg.
*               Bit di controllo PRB_SAVEIO.
*               Impostazione dell'handle di output per il
                figlio.
*               Altre tecniche
*
*/

#include <exec/types.h>
#include <libraries/arpbase.h>
#include <arpfunctions.h>

/* Il PCB di stato */

Struct ProcessControlBlock Status_PCB =
{
  10240,
  1,                    /* Questo processo ha la priorita' */
};

/* Il PCB per Type */

Struct ProcessControlBlock Type_PCB =
{
  10240,
  -1,           /* Questo processo ha una priorita'
                                                inferiore */
};

/* Il messaggio che vogliamo sia mandato quando il
nostro figlio muore.
* Inizializzato nel corpo del programma.
*/
struct ZombieMsg Status_Zombie, Type_Zombie;

/* Una struttura che viene attaccata al messaggio
Zombie per ricevere
*  piu' informazioni.
*/
struct MyUserInfo
{
  LONG  Return;         /* Ritorno iniziale di AsyncRun() */
  BYTE *Name;                   /* Nome del comando */
  struct DateStamp StartTime;  /* Tempo di inizio (ap
                                        prossimativo) */
} StatusInfo, TypeInfo;

/* Variabili per la conversione di data e ora */

BYTE    Day[LEN_DATSTRING],
        Date[LEN_DATSTRING],
        Time[LEN_DATSTRING];

struct DateTime dt =
{
  0,0,0,                        /* Il datestamp */
  FORMAT_DOS,                   /* per il momento */
        0,
        Day,
        Date,
        Time,
};

VOID
        main()
{
        ULONG Output(), Open(), File = 0;
        struct MsgPort *RendevousPort = NULL;
        register int messages_pending = 0;
        VOID Report_Stats();

        if ( !(RendevousPort = CreatePort(0L, 0L)) )
        {
                Puts("Non posso creare la
                Message Port!");
                exit(1);
}

/* Notate che questo file viene chiuso dal figlio, anche
                                                se non viene
* eseguito, cioe' se ASyncRun() restituisce un codice
```

```
                                                    d'errore.
*/
        if ( !(File = Open("TEST_FILE",
                                        MODE_NEWFILE)) )
{
                Puts("Non posso creare il file di test!");
                exit(1);
}

/* Bisogna inizializzare la porta di risposta del messaggio
                                                    Zombie */
Status_Zombie.zm_ExecMessage.mn_ReplyPort =
  Type_Zombie.zm_ExecMessage.mn_ReplyPort =
                                        RendevousPort;

/* Informiamo ASyncRun che vogliamo una risposta */
Status_PCB.pcb_LastGasp = &Status_Zombie;
Type_PCB.pcb_LastGasp = &Type_Zombie;

/* Usiamo una struttura custom per registrare il nome del
                                                    processo,
* il suo stato e l'ora di inizio attivita'
*/
StatusInfo.Return = TypeInfo.Return = -1;
StatusInfo.Name = "Status";
TypeInfo.Name = "Type";

Status_Zombie.zm_UserInfo = (ULONG)&StatusInfo;
Type_Zombie.zm_UserInfo = (ULONG)&TypeInfo;

/* Infine permettiamo che status usi il nostro handle di
                                                    output */
Status_PCB.pcb_Output = Output();

/* Non vogliamo che il programma status chiuda il nostro
                                                    handle
* cosi' facciamo:
*/
Status_PCB.pcb_Control |= PRF_SAVEIO;

/* Type usera' il file di test e lo chiudera' quando ha finito
*/
Type_PCB.pcb_Output = File;

/* L'impostazione e' terminata, far partire i processi.
* Notate che aspetteremo SOLO i processi che sono
                                        partiti con successo
* tramite ASyncRun. Se c'e' un problema, ASyncRun()
                                        restituisce un
* valore negativo.
*/

DateStamp( &TypeInfo.StartTime );       /* ricava l'ora di
                                        partenza */

if ( (TypeInfo.Return = ASyncRun("Type", "Sync.c Opt N",
                                &Type_PCB)) >= 0)
{
        messages_pending++;

DateStamp( &StatusInfo.StartTime );
if ( (StatusInfo.Return = ASyncRun("Status", "Full",
                                &Status_PCB)) >=0)
                        messages_pending++;

while (messages_pending)
{
  /* Lascia che i figli vengano eseguiti in background */
                        WaitPort(RendevousPort);
                                        GetMsg(RendevousPort);
                                        messages_pending--;
                }
        }

/* Ora stampa le statistiche */
        Report_Stats( &Status_Zombie );
        Report_Stats( &Type_Zombie );
        DeletePort(RendevousPort);
        exit(0);
}

/* Stampa le statistiche di conclusione di un processo */

VOID Report_Stats( z )
        struct ZombieMsg *z;
{
        register struct MyUserInfo *u;

        u = (struct MyUserInfo *)z->zm_UserInfo;

        if ( u->Return < 0)
        {
        Printf("Il processo %s non e' stato eseguito:
                Errore = %ld\n", u->Name, u->Return);
        return;
}
Printf(CLI %ld con nome %s ", z->zm_TaskNum, u-
                                                >Name);

/* Converte il datestamp in una forma leggibile */
dt.dat_Stamp = u->StartTime;

if ( !StamptoStr( &dt ) )
        Printf("Iniziato a %s del %s\n", dt.dat_StartTime,
dt.dat_StrDate);
else
        Puts("<oops><oops><oops>");

dt.dat_Stamp = z->zm_ExitTime;

if ( !StamptoStr( &dt ) )
        Printf("Uscito a %s del %s\n", dt.dat_StrTime,
dt.dat_StrDate);
else
        Puts("<oops><oops><oops>");

/* Se c'e' un errore, stampa il codice d'uscita e IoErr() */

if (z->zm_ReturnCode != 0)
{
        Printf("Processo terminato con stato = %ld, Il
                risultato = %ld\n",z->zm_ReturnCode, z-
                                                >zm_Result2);
        }
}
```

Listato 4

```
/* wbproc.c — Copyright (c) 1988 di Scott Ballantyne
*       Usate e abusate quanto volete
*
*       Programma che simula il Workbench.
*       Dimostra l'uso del bit PRB_NOCLI per creare un
        processo non-CLI
*       usando ASyncRun().
*/
```

```
#include <exec/types.h>
#include <libraries/arpbase.h>
#include <arpfunctions.h>
#include <workbench/startup.h>

struct ProcessControlBlock PCB =
{
  10240,
  0,
  PRF_NOCLI, /* Processo non-CLI */
};

struct WBStartup Fake; /* Il nostro messaggio
                                Workbench falso */

VOID
        main()
{
        struct MsgPort *WBPort;

if ( !(WBPort = Fake.sm_Message.mn_ReplyPort
                        = CreatePort( 0L, 0L)) )
{
        Puts("Non riesco a ottenere una message port!");
                exit(1);
}

if ( !(PCB.pcb_LoadedCode = Fake.sm_Segment
                        = LoadPrg("Clock")) )
{
        Puts("Non posso caricare il programma!");
        DeletePort( WBPort );
        exit(1);
}

/* Lancia il programma */

if ASyncRun("Clock", NOCMD, &PCB) >= 0 )
{
/* Imposta la message port del figlionel messaggio */
        Fake.sm_Process = PCB.pcb_WBProcess;

/* Il resto del massaggio, per questo programma, può
essere NULL */
        Fake.sm_NumArgs = 0;
        Fake.sm_ToolWindow = NULL;
        Fake.sm_ArgList = NULL;

/* Ora spediamo il messaggio, in modo che
                        il programma parta */
        PutMsg( Fake.sm_Process, &Fake );

/* A questo punto, si fa' normalmente qualcosa;
                        comunque se non c'e'
* nulla da fare aspetteremo ...
*/
        WaitPort( WBPort );
        GetMsg( WBPort );
}
else            /* ERRORE */
        Puts("Non posso lanciare il programma!");
/ PULIZIA: Normalmente si deve solo fare un
                DeletePort(), ma siccome
* abbiamo usato la variabile pcb_LoadedCode
                dobbiamo anche scaricare
* il programma.
*/

        UnLoadPrg( PCB.pcb_LoadedCode );

        DeletePort( WBPort );
}
```

Listato 5

```
/* magic.c — Copyright (c) 1988 di Scott Ballantyne
*       Usate e abusate quanto volete
*
*       Rende una parte di sé stesso un sotto-processo,
        usando ASyncRun()
*
*       PER GLI UTENTI LATTICE: compilate
        con -y e -v
*/

#include <exec/types.h>
#include <libraries/arpbase.h>
#include <libraries/dos.h>
#include <arpfunctions.h>

typedef void PROCESS;

PROCESS MyProcess();

struct ProcessControlBlock PCB =
{
        6000,           /* dimensione dello stack */
        0,              /* priorita' */
        PRF_CODE | PRF_NOCLI | PRF_SAVEIO
        /* flag di controllo */
,
};

struct ZombieMsg ChildMsg;      /* Dobbiamo
                        riceverlo prima di uscire */
VOID
        main()
{
        register struct MsgPort *RendevousPort = NULL;
        register LONG rc;
        register int i;

if ( !(RendevousPort = ChildMsg.zm_
        ExecMessage.mn.ReplyPort =
        CreatePort( 0L, 0L )) )
{
        Puts("Non posso creare la porta!");
        exit(1);
}

/* Chiediamo di essere informati all'uscita */
        PCB.pcb_LastGasp = &ChildMsg;

/* ASyncRun() punta al nostro codice */
        PCB.pcb_LoadedCode = (CPTR)MyProcess;

/* Attiva il sotto-processo */
if ( (rc= ASyncRun("MyProcess", NOCMD, &PCB)) >= 0)
{
/* Si puo' usare o meno la MsgPort del figlio come meglio
si crede,

* ma se si usa, bisogna fare attenzione quando il figlio
richiama i processi
* AmigaDOS.

* Notate che accade cio' poiché abbiamo impostato il bit
PRB_NOCLI
```

```
* PRB_CODE non lo fornisce automaticamente

        Printf("PADRE: la porta del figlio si trova a
        %08lx\n", PCB.pcb_WBProcess);

        for (i = 0; i < 300; i++)
                Printf("%d", i);
        Printf("PADRE: Finito di contare\n");

        WaitPort( RendevousPort );
        GetMsg( RendevousPort );
}
else
        Printf("PADRE: Non posso far partire il figlio,
        ERRORE = %ld\n", rc);
        Printf("PADRE: Il figlio e' uscito con stato  =
        %ld\n", ChildMsg.zm_ReturnCode);
        DeletePort(RendevousPort);
}

/* Di seguito è riportato il codice del processo figlio. Si
                                        può fare grafica,
* gestione asincrona dell'I/O o qualunque cosa si
                                        desideri. Si deve solo
* aprire un file, scrivere qualche cosa per provare che
                                        si sta usando
* qualche funzione DOS e uscire.
* Notate che non si può richiamare nessuna funzione
                                        di libreria C, visto
* che non siamo entrati in MyProcess usando la normale
                                        routine di startup.
* Ecco la ragione dell'uso della funzione Exit() di Amiga
                                        invece di exit().

PROCESS
        MyProcess()
{
        ULONG Open(), MyFile;
        register int i;
#ifdef AZTEC_C
        geta4();
#endif
        if ( !(MyFile = Open("CON:0/0/640/100/Finestra
                di IO del figlio", MODE_NEWFILE)))
                Exit(20L);

        for ( i = 0; i < 100; i++)
        FPrintf(MyFile, "%d  ", i);

        FPrintf(MyFile, "\nFIGLIO: Ho finito di contare
                                e sto uscendo\n");

/* Aspettiamo un po' in modo che si possa leggere il
                                        messaggio */
        Delay(60L)
        Close(MyFile);
        Exit(0L);
}
```

Listato 6

```
/*dodate.c - Conversione di date

/* dodate.c — Copyright (c) 1988 di Scott Ballantyne
*                       Usate e abusate quanto volete
*
*       Esempio di come si effettuano le canoniche
                                conversioni di date con
*       StamptoStr() e StrtoStamp(), usando anche la
        variabile d'ambiente
*       'dateformat'
*/
#include <exec/types.h>
#include <libraries/arpbase.h>
#include <arpfunctions.h>

/* Memoria per le stringhe */

BYTE EnvBuf[ LEN_DATSTRING ];
BYTE DayStr[ LEN_DATSTRING ];
BYTE DatStr[ LEN_DATSTRING ];
BYTE TimStr[ LEN_DATSTRING ];

/* Sia StamptoStr() che StrtoStamp() usano la seguente
                                        struttura per
* controllare il formato dell'input e dell'output */
struct DateTime MyDT =
{
        0, 0, 0,  /* Una struttura DOS DateStamp */
        0, 0,     /* Formato e flag, vedremo piu' avanti */
        &DayStr[0],
        &DatStr[0],
        &TimStr[0],
};

BYTE *Prompts[] =
{
        "DD-MMM-YY, cioe' 23-Jan-88",
        "YY-MM-DD, cioe' 88-01-23",
        "MM-DD-YY, cioe' 01-23-88",
        "DD-MM-YY, cioe' 23-01-88",
};

BYTE Buffer[ MaxInputBuf ];

VOID
        main()
{
        register char *cp = EnvBuf;

/* Per prima cosa determina i gusti dell'utente */
        if ( Getenv( "dateformat", EnvBuf,
                                LEN_DATSTRING ) )
        {
/* L'utente ha espresso una preferenza, quindi dobbiamo
                                        interpretare cio'
                che vuole. Cerchiamo un '-' iniziale. */
        if ( *cp == '-' )
                {
/* Sicuramente e' un utente smaliziato, visto che non
                                vuole la conversione
                  delle date: quindi disabilitiamola. */
                MyDT.dat_Flags |+ ( ~DTF_SUBST );

/* Avanza oltre il '-' per Atol() */
                        cp++;
}
else
                MyDT.dat_Flags |= DTF_SUBST;

/* Estrae il formato. Atol() restituisce 0 per le cifre
                                        scorrette e non
        considera valori superiori a FORMAT_MAX*/
                MyDT.dat_Format = Atol( cp );
}
else
                MyDT.dat_Format = FORMAT_DOS;
```

```
                                /* Usa il default */

        MyDT.dat_Flags |= DTF_FUTURE;

do
{
        Printf("Inserite la data (%s): ",
                Prompts[ (MyDT.dat_Format >
        FORMAT_MAX) ? 0 : MyDT.dat_Format]);

        ReadLine( Buffer );
        /* Copiamo nella posizione corretta in MyDT */
        strncpy( MyDT.dat_StrDate, Buffer, 10);
        Printf("Inserite l'ora (OO:MM:SS): ");
        ReadLine(Buffer);
        strncpy( MyDT.dat_StrTime, Buffer, 10);
}
while ( StrtoStamp( &MyDT ) != 0 );
        Printf(" Ho convertito i valori in %ld Giorni, %ld
Minuti e %ld Tick\n",
        MyDT.dat_Stamp.ds_Days,
        MyDT.dat_Stamp.ds_Minute,
        MyDT.dat_Stamp.ds_Tick );

/* Per le altre funzioni, leggiamo solamente l'ora corrente
                                        del sistema,
                la convertiamo e la visualizziamo. */
DateStamp( (struct DateStamp *)&MyDT );
                        /* Ottiene l'ora del sistema */

if ( StamptoStr( &MyDT ))
        Puts("<oops><oops><oops>");
else
{
        Printf("L'ora corrente del sistema e' %s %s\n",
                MyDT.dat_StrDate, MyDT.dat_StrTime );
}
/* FINE */
}
```

Listato 7

```
/*freq.c - Uso del file requester

/* freq.c — Copyright (c) 1988 di Charlie Heath
*               Usate e abusate quanto volete
*
*               Illustra:
*               Il semplice uso di FileRequest()
*/

#include <exec/types.h>
#include <libraries/arpbase.h>
#include <Intuition/intuition.h>

#ifdef AZTEC_C
#include <functions.h>
#include <arpfunctions.h>
#else
#include <proto/arp_pragmas.h>
#include <proto/intuition.h>
#endif

/* La struttura dati FileRequester, che e' passata come
argomento del
* chiamante, ha bisogno di due buffer di caratteri per
FILENAME e
* DIRECTORYNAME
*/
char            FileName[FCHARS+1];
char            DirName[DSIZE+1];

struct FileRequester MyFileRequest =
        {
        "Una mia stringa",              /* Prompt */
        FileName,
                /*Puntatore al buffer per il filename */
        DirName,
            /* Idem come sopra per il directoryname */
        0,
                        /* Finestra oppure NULL */
        0,0,
                        /* Flag di controllo speciali */
        0,
                /* Funzione Speciale (qui NULL) */
        0
                                        /* Riservato */
        };

IMPORT struct Arpbase *ArpBase;

void main(argc,argv)
int             argc;
char            *argv[];
{

/* Il primo esempio e' semplice: bisogna usare la
                                        struttura gia' pronta
* definita appena sopra e chiamare FileRequest.
* Questo e' cio' che si deve fare se non si usa
                                        una finestra e si vuole che
* appaia un file requester sullo schermo del Workbench.
*/

Printf("Per prima cosa apriamo il file requester
                                        senza finestra...\n\n");
if ( FileRequest( &MyFileRequest) == 0L )
        {
        Printf("L'utente ha cancellatto il file
                                                requester!!!\n");
        }
else
        {
        Printf("L'utente ha selezionato il FILE: %s e la
        DIRECTORY: %s\n",FileName,DirName);
                }
/* L'esempio successivo e' piu' complesso e quindi
                                        l'abbiamo messo in
                                        una funzione separata */

Printf("Poi facciamo qualcosa di meglio...\n\n");
DoFancyStuff();
}

/* Il resto del codice e dei dati e' un esempio di come
                                        si possono usare
* molte delle personalizzazioni per il file requester
*/

/* Per prima cosa abbiamo bisogno di uno schermo
CUSTOM con una finestra aperta */

struct TextAttr MyTextAttr = { (UBYTE*)
                "topaz.font",TOPAZ_EIGHTY,0,0 };

struct NewScreen MyNewScreen =
        {
```

```
        0,50,320,200,3,
        1,2,
        0,CUSTOMSCREEN,&MyTextAttr,
        (UBYTE *)"Il mio schermo",
        0,0
        };

struct NewWindow MyNewWindow =

        {
        10,10,300,150,
        1,2,
        REFRESHWINDOW+NEWSIZE+ACTIVEWINDOW,
        ACTIVEWINDOW+SMART_REFRESH+WINDOWDEPTH
        +WINDOWSIZING+WINDOWDRAG,
        0,0,
        (UBYTE *)"La mia finestra",
        0,
        0,
        20,20,1000,1000,
        CUSTOMSCREEN
        };

/* aggiungiamo questo gadget */

#define MYGADGETID          1
struct IntuiText drtb = {1,2,JAM2, 88, 3, &MyTextAttr
        (UBYTE *)"Gadget personale!",NULL};

struct Gadget srg3 = { NULL , 5, 128, 310, 14,
        GADGHCOMP, RELVERIFY | TOGGLE
        SELECT, BOOLGADGET,
        NULL, NULL, &drtb, NULL, NULL, MY
        GADGETID, NULL };

/* Lo scopo della funzione seguente e' di trattare gli
                                        eventi speciali
* impostati con fr_Flags all'interno della chiamata
                                        a FileRequest.
*/

MyFunc(Mask,Object)
ULONGMask
CPTR            *Object;

{
switch ( Mask )
        {
        case FRF NewWindFunc:
/* Bisogna fare qualcosa del genere se si vuole
                        usare FileRequest() in uno
                                schermo LORES */
                ((struct NewWindow *)Object)->
                                        LeftEdge = 0;
                ((struct NewWindow *)Object)->
                                Height = 128 + 15;
                break;
        case FRF_AddGadFunc:
                AddGadget( (struct Window
                                *)Object),&srg3, 0L);
                break;

        case FRF_GEventFunc:

        Printf("Richiamata la funzione Gadget Event\n");
/* Questo codice di ritorno, purtroppo, non funziona
                                con ARP 1.1 */
                return(TRUE);
        case FRF_DoMsgFunc:
                Printf("Richiamata la funzione Message
                                        Event\n");
                ReplyMsg( Object );
                break;
                }
}

/* Questa funzione esegue tutti quei compiti semplici
                                come aprire lo schermo,
* la finestra, impostare i flag speciali, ecc.
*/
DoFancyStuff()
{
struct Screen           *MyScreen;
struct Window           *MyWindow;

ULONGretval;

MyScreen = OpenScree(&MyNewScreen);
if ( MyScreen == 0L )
        {
        Printf("Non posso aprire lo schermo!\n");
        return(1);
        }
MyNewWindow.Screen = MyScreen;
MyWindow = OpenWindow(&MyNewWindow);
if ( MyWindow == 0L )

        {
        Printf("Non posso aprire la finestra!\n");
        return(2);
        }

/* Ora abbiamo uno schermo con una bella
                                finestra aperta...
* Bisogna quindi impostare i flag speciali che si
                                desiderano usare.
* Notate che si sono lasciati la maggior parte dei valori
                                di default nella
* struttura FileRequester.
*/

MyFileRequest.fr_Function = (VOID (*)())MyFunc;
MyFileRequest.fr_Window  = MyWindow;
MyFileRequest.fr_FuncFlags = FRF_AddGad
                        Func+FRF_GEventFunc+
                        FRF_NewWindowFunc+FRF_
                        DoMsgFunc;
retval = (ULONG)FileRequest( &MyFileRequest );

CloseWindow(MyWindow);
CloseScreen(MyScreen);

if ( retval == 0L )
                {
                Printf("L'utente ha cancellato il file
                requester nello schermo!\n");
                }
else
                {
                Printf("(Nel CUSTOMSCREEN...)\n");
                Printf("L'utente ha scelto il FILE: %s e la
                DIRECTORY:%s\n", FileName,
                DirName);

}
```

Come cambiare il "mouse pointer" in AmigaBasic

di Anthony Bryant

Un breve programma per cambiare il "mouse pointer"di default con uno di propria creazione.

Questo articolo si rivolge agli amanti del Basic che vogliono usare un pointer personalizzato nei loro programmi o agli estimatori del C o dell'assembler che vogliono provare rapidamente e in maniera interattiva un nuovo pointer (ovvero un nuovo sprite). Con il comando LIBRARY dell'AmigaBasic, è possibile accedere a funzioni di sistema altrimenti non utilizzabili tramite i normali comandi del Basic. Le funzioni di Intuition SetPointer() e ClearPointer() verranno, quindi, usate nel programma descritto in questo articolo per passare dal pointer di sistema a quello personalizzato e viceversa.

Come si usano le librerie in AmigaBasic

Il pointer a freccia è quello usato per default da Intuition, per cui per cambiarlo (il pointer è in termini di hardware lo sprite 0) dobbiamo servirci di Intuition. Le funzioni di Intuition possono essere chiamate, come del resto tutte le funzioni delle altre librerie, tramite il comando LIBRARY. Questo comando rende disponibili tutte le funzioni della libreria il cui nome è specificato dalla stringa che lo segue (in questo caso sarebbe "intuition.library").
Il problema, però, è che affinché il comando LIBRARY possa funzionare, deve essere disponibile il file ".bmap" associato alla library che si intende usare. Tale file contiene, infatti, le necessarie informazioni relative alle funzioni della libreria che il Basic deve conoscere, come il loro indirizzo in memoria e il numero e il tipo di parametri che esse richiedono.

Poiché l'informazione contenuta in un file ".bmap" si riferisce specificamente a una particolare versione del sistema operativo, il file deve essere generato ad hoc per quella che si usa abitualmente.
Il programma "ConvertFD" nel drawer "BasicDemos" del disco dell'AmigaBasic (versione 1.2) creerà un file ".bmap" da un file ".fd", file che contiene informazioni più generali riguardo le funzioni di una libreria. I file ".fd" relativi a tutte le library possono essere reperiti nella directory "fd1.2" del disco dell'AmigaBasic. Tornando al nostro caso, il file "intuition.bmap" potrà essere creato con il comando:

```
ConvertFD :fd1.2/intuition_lib.fd
```

Ci sarà bisogno anche del file ".bmap" relativo alla exec.library, per cui usate "ConvertFD" anche sul file "exec_lib.fd", anch'esso nella directory "fd1.2".
Potrebbe risultare utile tenere questi file a portata di mano per i futuri programmi: sarebbe allora una buona idea creare una directory "libs" sul disco che usate per l'AmigaBasic per porvi tutti i vostri file ".bmap". In tal caso, per usarli con il comando LIBRARY si deve specificare l'intero path dei file, ovvero il nome per esteso come:

```
LIBRARY ":libs/intuition.library"
LIBRARY ":libs/exec.library"
ecc.
```

questo partendo dal presupposto che la vostra directory corrente si trovi da qualche altra parte sullo stesso disco.

Una volta "collegati" alla libreria di sistema prescelta per mezzo del comando LIBRARY, potete chiamare le funzioni con il loro nome e passare loro gli argomenti proprio come in un programma C. Le routine che restituiscono un valore devono essere dichiarate con:

```
DECLARE FUNCTION NomeDiFunzione LIBRARY
```

usando un "&" al termine del nome della funzione nel caso in cui essa restituisca un "long", cioè un intero a 32 bit, cosa che per inciso fanno la maggior parte delle funzioni di sistema.

Probabilmente avrete già esaminato i vari programmi dimostrativi che impiegano il comando LIBRARY per rendervi conto di come funziona e magari avrete già dato un'occhiata ai file "fd1.2" per familiarizzare con i nomi di quelle funzioni che siete così ansiosi di provare. Sappiate, comunque, che i manuali (quello di Intuition e il ROM Kernel) sono assolutamente necessari per servirsi proficuamente delle librerie di sistema.

Il manuale dell'Intuition

Il capitolo 4 del manuale di Intuition tratta delle funzioni preposte alla creazione e all'impiego di un pointer personalizzato in una

finestra:

```
SetPointer(Window,Pointer,height,width,xoffset,yoffset)
```

cambia il pointer nella finestra specificata e

```
ClearPointer(Window)
```

vi ripristina il pointer di sistema.

Il programma che segue è una semplice traduzione in AmigaBasic. Si tratta fondamentalmente di passare a SetPointer() i giusti argomenti e poi di chiamare ClearPointer() per ripristinare il pointer di default. Il puntatore alla finestra viene ottenuto con la funzione WINDOW(7) e i rimanenti argomenti di SetPointer() fanno parte della definizione dello sprite contenuta nelle linee DATA.

Passare il puntatore ai dati di definizione dello sprite a SetPointer è un po' più complicato.
Forse il metodo più semplice sarebbe quello di immagazzinare la definizione dello sprite in un array di interi e passare a SetPointer() un puntatore all'array tramite la funzione del Basic VARPTR. Il problema di questo approccio è che le definizioni di sprite, come tutti i dati che devono essere accessibili all'hardware che si occupa della grafica e del suono, devono essere posti nella CHIP RAM. Un Amiga senza espansione di memoria ha solamente ram di tipo CHIP, per cui l'uso dell'array non dà problemi, ma se venisse aggiunta un espansione il programma non funzionerebbe più del tutto.
Per operare in maniera totalmente corretta, la memoria viene allocata con la funzione AllocMem() della libreria Exec; per mezzo di tale funzione, è possibile richiedere espressamente solo memoria di tipo CHIP. La definizione dello sprite viene trasferita nella zona di memoria allocata con POKEW (POKE Word) e un ciclo FOR...NEXT.
Prima che il programma finisca, la memoria allocata viene restituita al sistema con FreeMem(). Se questo non venisse fatto, la memoria allocata sarebbe persa fino al prossimo reset della macchina.
Per usare le funzioni AllocMem e FreeMem, deve essere usato il comando LIBRARY per la "exec.library".
Inoltre, AllocMem restituisce un "long" come risultato edi conseguenza deve essere esplicitamente dichiarato aggiungendo, come visto in precedenza, un "&" nell'istruzione DECLARE FUNCTION.
Quando eseguirete il programma (la finestra "run" del Basic deve essere attiva), vi apparirà il pointer nuovo fino a quando non premerete il tasto sinistro del mouse.
A questo punto verrà ripristinato il pointer di sistema (la freccia o qualsiasi altra cosa abbiate impostato con Preferences) e il programma terminerà. Certamente questo programma non ha molta utilità di per se stesso, ma fornisce uno spunto di studio e un modo per usare nei vostri programmi un pointer personalizzato.
Potete variare il comando RESTORE per provare ognuno dei 4 pointer definiti nelle linee DATA o provare addirittura a cambiare i dati per creare voi stessi nuovi pointer.

Ad ogni modo, per creare un pointer ex novo è necessaria una certa conoscenza di come funzionino gli sprite.

Come creare uno sprite

L'insieme dei dati che costituiscono la definizione di uno sprite è composta da due word (interi a 16 bit) per linea, riguardanti informazioni di posizione e colore; in realtà, le prime e le ultime due parole non descrivono un bel niente e devono essere poste a zero poiché vengono usate dal sistema per sue esigenze specifiche.
Ogni gruppo di due word descrive il colore di ogni pixel di una linea orizzontale dello sprite. Una word è composta da 16 bit e 16 bit è la massima larghezza di uno sprite. La prima word contiene i bit meno significativi e la seconda quelli più significativi per ogni singolo pixel dello sprite; i due bit per pixel individuano uno tra quattro possibili colori:

	msb	lsb	
Colore	0	0	è trasparente
Colore	0	1	è quello di media intensità (rosso - registro di colore 17)
Colore	1	0	è quello di bassa intensità (nero - registro di colore 18)
Colore	1	1	è quello di alta intensità (rosso - registro di colore 19)

Nel programma vengono presentati alcuni esempi di pointer appositamente creati per mostrare come funzionano i colori (i dati sono in esadecimale). XPointer è una copia fedele dell'esempio in C presente nel manuale di Intuition; XPlain è la versione a un solo colore.

```
'** Un pointer diverso da quello di default **

DECLARE FUNCTION AllocMem& LIBRARY
LIBRARY "intuition.library"
LIBRARY "exec.library"

chip% = 2  'per richiedere con AllocMem esclusivamente chip ram

RESTORE Xpointer  'il pointer desiderato
READ pHeight%,pWidth%,pXOffset%,pYOffset%
pSize& = 2*(2*pHeight% + 3)  'numero di bytes da allocare per lo sprite
mem& = AllocMem&(pSize&,chip%) 'allocazione in chip ram

'** copia i dati dello sprite in chip ram **

FOR i& = mem& TO (mem& + pSize&) STEP 2
  READ x%
  POKEW i&,x%
NEXT i&
```

(segue a pagina 67)

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 _____

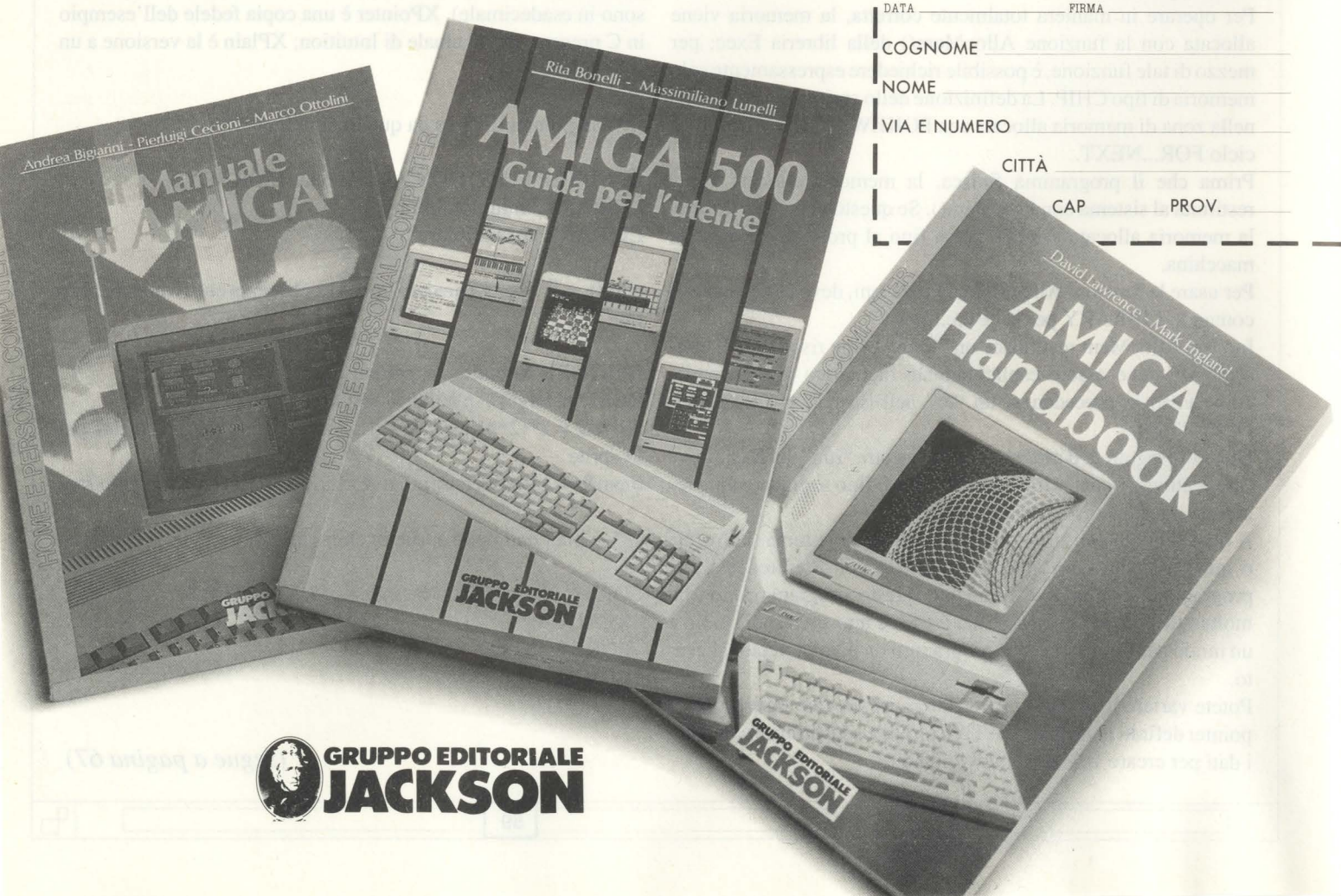
COGNOME _____

NOME _____

VIA E NUMERO _____

CITTÀ _____

CAP _____ PROV. _____



GRUPPO EDITORIALE JACKSON

Lo standard ANSI

di Eric Giguere

Un sommario per il programmatore in C

Eric Giguere ha partecipato alla realizzazione di un compilatore conforme allo standard ANSI al Computer System Group della università di Waterloo, presso la quale frequenta il terzo anno di informatica nel programma co-op.

I linguaggi di programmazione si stanno costantemente evolvendo e diversificando. Il linguaggio C non è una eccezione a questa regola, sopratutto a causa della sua crescente popolarità durante gli ultimi anni. Il documento originale di specifica per il C, *The C Programming Language* di Brian Kernighan e Dennis Ritchie (noto comunemente come il K&R) è ormai quasi vecchio di dieci anni. Il K&R è stato usato come la 'bibbia' dei programmatori in C, lo standard de facto per il linguaggio. Ma essendo il linguaggio in rapida evoluzione, diventa evidente la necessità di uno standard formale del linguaggio.

Sotto gli auspici della International Standars Organization (ISO), l'American National Standards Institute (ANSI) cominciò la preparazione di uno standard sotto il controllo del X3J11 Technical Committee. Lo standard proposto è ora nella forma di stesura finale e si aspetta la sua approvazione da parte dell'ANSI e dell'ISO per la fine di questo anno.

Questo articolo è un sommario dei maggiori cambiamenti attuati dallo standard proposto al linguaggio C ed è rivolto al programmatore già asperto.
Tutte le informazioni sono derivate dai documenti ufficiali del X3J11: *Il Draft Proposed American National Standard for Information Systems - Programming Language C*, e l'allegato *Rationale for Draft Proposed National Standard for Information System - Programming Language C*. Queste pubblicazioni verranno indicate in questo articolo rispettivamente come lo *Standard* e il *Rationale*.

L'articolo non è una critica allo *Standard* ma solo un commento. I lettori devono comunque stare attenti alla possibilità che vengano apportate delle variazioni allo *Standard* prima della sua approvazione finale da parte dell'ANSI.

Pochi programmatori hanno l'interesse o il tempo di spaziare attraverso il testo stesso della bozza dello *Standard* nella sua totalità. Ciò che li interessa sono le variazioni apportate dallo *Standard* a quello che era stato definito nel K&R e come questo si ripercuota sui loro programmi attuali. Questo articolo, essendo un sommario di questi cambiamenti, fornisce un riferimento veloce da consultare, che il programmatore C medio può leggere e comprendere in una sola volta.

Che cosa ci si aspetta dallo *Standard*

Lo *Standard* non vuole creare una nuova definizione di un linguaggio. Il suo scopo, per citare il *Rationale*, è di 'codificare una pratica comune'. Questo significa che la struttura fondamentale e la sintassi del linguaggio descritta nel K&R è stata lasciata intatta. Lo *Standard* ha, invece, cercato di unificare le diverse estensioni e i dialetti ('la pratica comune') sviluppatisi durante gli anni, in una singola definizione coesiva del linguaggio.
La pratica attuale è a volte inconsistente e si sono dovuti spesso fare dei compromessi. Forse la cosa più importante da ricordare sullo *Standard* è che non è stato pensato per invalidare il codice C esistente. I programmi già realizzati potranno ancora essere compilati, effettuando solamente delle piccole modifiche, da compilatori conformi allo *Standard* ANSI.

Identificatori riservati

Sono state aggiunte al linguaggio le seguenti parole chiave:

const, enum, signed, volatile

La spiegazione per ognuna di esse segue più avanti. In aggiunta 'identifier' è stato tolto dalla lista delle parole riservate così come non era mai stata implementata dal K&R o dalle successive versioni del C.

Tipi di dati

Le maggiori modifiche al linguaggio riguardano i tipi di dati.

Interi: La lista dei tipi di interi è stata ampliata per includere i 'signed char' e le seguenti variazioni:

unsigned int
signed int
unsigned long
signed long
unsigned short
signed short

I tipi 'long int' e 'short int' possono essere usati come varianti di 'long' e 'short' rispettivamente. Le dichiarazioni:

signed x; unsigned y;

possono essere usate come abbreviazione per 'signed int' e

'unsigned int'. Da notare che deve essere presente almeno una classe di memorizzazione ('auto','register',static', 'extern') o uno specificatore di tipo quando si dichiara una variabile. Una dichiarazione del tipo *x*; non è più ammessa e deve essere sostituita con int *x*; per poter essere compilata.

Se un semplice 'char' debba essere considerato o no 'signed' oppure 'unsigned' è lasciato al particolare compilatore.

Float: è stato aggiunto un nuovo tipo chiamato 'long double' per poter avere una maggiore precisione, ma come qualunque tipo 'long', si garantisce solamente che sia almeno grande quanto il tipo inferiore, in questo caso un 'double'. Il tipo 'long float' (un sinonimo per 'double') non è più valido.

Strutture e unioni: Le aree di allocazione dei nomi dei vari elementi sono ora uniche. Ciò significa che due diverse strutture o unioni possono contenere elementi che abbiano lo stesso nome senza possibilità di generare un conflitto. Stutture e unioni possono ora:

- essere assegnate a un'altra dello stesso tipo.
- essere inizializzate quando vengono dichiarate di tipo 'auto'.
- essere passate come parametri di funzioni e valori di ritorno.

Enumerazioni: Già disponibili nella maggior parte dei compilatori, le enumerazioni sono state aggiunte ufficialmente al linguaggio. Una enumerazione è il modo di dichiarare un insieme di costanti intere. La dichiarazione:

```
enum colours {RED,BLUE,GREEN};
```

dichiara 'colours' come un insieme enumerativo che rappresenta le costanti intere RED, BLUE e GREEN. Queste costanti enumerative sono dei valori interi che partono da 0 e sono incrementati di 1 per ogni identificatore. Ciascuna di esse ha come tipo 'int' e può essere usata come tutti gli altri interi. Usarla è l'equivalente di scrivere:

```
#define RED     0
#define BLUE    1
#define GREEN   2
```

Possono essere dichiarate delle variabili in modo che assumano tipo enumerativo:

```
enum colours x,y;
```

dichiara x e y come interi capaci di contenere una costante enumerativa di tipo 'colours'. In pratica, non viene generato alcun controllo per essere sicuri che vengano utilizzate solamente costanti enumerative, così che i seguenti assegnamenti risultano equivalenti:

```
x = BLUE;
x = 1;              /* annulla lo scopo */
```

Le costanti enumerative non devono essere forzatamente in maiuscolo, ma il maiuscolo è una convenzione riconosciuta per le costanti. I valori costanti possono essere assegnati direttamente all'interno di una enumerazione:

```
enum relation {
        EQUAL = 1 , LESS_THAN = 2 ,
        GREATHER_THAN = 4 };
```

Se non si specifica un valore per un dato identificatore gli viene attribuito quello della costante che lo precede incrementato di uno.

Tipo void: è stato aggiunto il tipo di dato 'void' per indicare che una espressione non ritorna alcun valore. Non si può dichiarare nessuna variabile con questo tipo, ma si può fare un cast di tipo void su delle espressioni. Per esempio la seguente dichiarazione:

```
(void)printf("Hello world\n");
```

indica in modo specifico al compilatore che il valore di ritorno della 'printf' (un intero) deve essere ignorato. Come tale, l'istruzione seguente è illegale:

```
a = (void)func();  /* illegale */
```

perché l'operatore di assegnazione aspetta una variabile di ritorno per completare la sua funzione.

I puntatori 'void' e le funzioni 'void' sono discussi subito dopo e nel paragrafo 'Dichiarazioni di funzioni'.

Puntatori e vettori: I puntatori non sono più sinonimi di 'int' e possono solo essere confrontati con, o assegnati a:

- il valore intero 0 (usato per definire un puntatore 'null').
- un puntatore dello stesso tipo.
- un puntatore di un tipo generico (un puntatore 'void').

Un puntatore 'void' è un puntatore che non ha un tipo base (cioè punta a un tipo di dimensioni non conosciute) ed è dichiarato usando la sintassi:

```
void *ptr;
```

Non è ammessa l'indirezione attraverso un puntatore 'void'; deve prima essere fatto un cast a un tipo di puntatore appropriato.

I vettori con classe di memorizzazione 'auto' possono ora essere inizializzati e, se specificata, la dimensione di un vettore deve essere una espressione intera maggiore di zero.

Modificatori speciali: Lo *Standard* mette a disposizione i due attributi 'const' e 'volatile' che possono essere usati come modificatori di tipo.

Un oggetto dichiarato come 'const' non può essere modificato (assegnato, incrementato, decrementato) da un programma. Quindi il codice seguente non è valido:

```
const int x;
/* ... */
x = 2;              /* illegale */
```

Invece è possibile farne l'inizializzazione:

```
const unsigned int mask[] = {
        0x000 , 0x00ff , 0xff00 , 0xffff };
```

Un oggetto 'const' (se è 'static') può essere un dato allocato in memoria read-only (ROM). Dichiarando questo dato come 'const' si permette al compilatore di effettuare una diagnostica su ogni (inutile) tentativo di modificarne il contenuto.
Il tipo 'volatile' può essere modificato al di fuori del controllo del programma. Locazioni di I/O mappate in memoria ne sono dei tipici esempi. Dichiarando un oggetto come 'volatile' si indica che il compilatore deve sempre estrarne il valore dalla sua locazione in memoria e che questo valore può essere variato dal momento dell'ultimo accesso fatto.
Questo previene le possibili ottimizzazioni che tenterebbero di inserire il valore in un registro della CPU generando dei risultati non corretti.

```
volatile char *port1 = 0x00f3;    /* puntatore a una
                                     porta di I/O */
while(*port1 & DATA_FLAG)         /* deve essere di
                                     tipo volatile */
    clear_io();
```

I modificatori 'const' e 'volatile' possono essere usati (singolarmente o insieme) in combinazione con gli altri specificatori di tipo validi.

Anche i puntatori possono essere dichiarati 'const' o 'volatile' attraverso l'uso di una particolare sintassi che viene compresa meglio attraverso un esempio:

```
cost int *x;
int *const y;
```

Nell'esempio x è dichiarato come puntatore a una costante intera, mentre y è dichiarato come un puntatore costante a un intero. La distinzione consiste nel dove mettere l'attributo 'const'. Consideriamo le seguenti istruzioni:

```
x = NULL;       /* corretto */
*x = 0;         /* errato */
y = NULL;       /* errato */
*y = 0;         /* corretto */
```

Poiché x è un puntatore a un 'const int', il valore a cui punta non potrà mai variare. Analogamente, poiché y è un puntatore 'const' a un 'int' non potrà mai essere modificato, sebbene sia modificabile il valore a cui punta. Regole simili esistono per l'attributo 'volatile', il quale può sempre essere usato insieme a 'const'.

Campi di bit: I campi di bit possono essere di tipo 'int', 'unsigned int' o 'signed int'. Il fatto che si consideri o meno un bit di segno il bit più significativo di un campo di bit dipende dall'implementazione.

Definizioni vuote: è ora ammessa una definizione vuota che consista solamente in uno specificatore di 'struct' o di 'union' con un nome associato. Il suo scopo è di nascondere qualsiasi dichiarazione esterna, con lo stesso nome, nel blocco corrente:

```
struct a
  {
        int x;
};

int func()
  {
        struct a st1;       /* struttura definita sopra */

        struct a;           /* definizione vuota */

        struct b
          {
                struct a *y; /* si riferisce alla
                                prossima struttura */
          } st2;

        struct a
          {
                struct b *z;
          } st3;

        st1.x = 1;
        st2.y = &st3;       /* &st1 avrebbe
                               generato un warning */
        st3.z = &st2;
  }

/* Questa è la visibilità della prima 'struct a' */
```

Nell'esempio l'elemento y di st2 è un puntatore a una seconda 'struct a', la quale è definita sotto di esso. Se l'istruzione 'struct a;' iniziale non fosse presente, y sarebbe stato un puntatore alla 'struct a' definita nel precedente livello di visibilità (scope).

Conversioni e promozioni: Le promozioni intere sono definite nel seguente modo: i tipi 'char', 'short' o i campi di bit, con o senza i modificatori 'signed' o 'unsigned' possono essere usati ovunque sia richiesto un 'int'. I valori verranno convertiti, se possibile, in interi; altrimenti in 'unsigned int'.
Le usuali conversioni aritmetiche usate con la maggior parte degli operatori binari sono state modificate in modo da riflettere la nuova disponibilità di tipi per il programmatore. Le conversioni sono applicate agli operandi di un'espressione nel seguente ordine:

- Per primo, se uno di essi è 'long double', l'altro è convertito in 'long double'.
- Altrimenti, se uno è 'double', l'altro è convertito in 'double'.

- Altrimenti, se uno è 'float', l'altro è convertito in 'float'.

- Altrimenti, vengono effettuate delle promozioni a interi. Quindi se un operando è 'unsigned long int', l'altro viene convertito in 'unsigned long int'.
- Altrimenti, se uno è 'long int' e l'altro è 'unsigned int', se un 'long int' può rappresentare tutti i valori di un 'unsigned int', allora un 'unsigned int' è convertito in 'long int', altrimenti entrambi sono convertiti in un 'unsigned long int'.

- Altrimenti, se uno è un 'long int', l'altro è convertito in 'long int'.

- Altrimenti, se uno è 'unsigned int', l'altro è convertito in 'unsigned int'.

Altrimenti, entrambi gli operandi devono avere come tipo 'int', come richiesto dalle promozioni a interi.

Lo *Standard* specifica anche altre regole che riguardano le conversioni da un 'unsigned' a 'signed', da interi a virgola mobile e da virgola mobile a virgola mobile. Queste regole sono state lasciate, per la maggior parte, identiche a quelle del K&R o sono (si suppone) senso comune dei programmatori.

Limiti minimi dei tipi: Ogni compilatore che segua lo Standard deve anche rispettare i seguenti limiti che riguardano l'intervallo di valori che ogni particolare tipo può accettare. Notate che questi sono limiti inferiori: una particolare implementazione è libera di eccedere per ciascuno di essi. Notate inoltre che il minimo intervallo di valori per un 'char' dipende dal fatto che il 'char' sia considerato o meno 'signed' oppure 'unsigned'.

- Numero di bit per un char	8
- Intervallo per un signed char	da -127 a +127
- Intervallo per un unsigned char	da 0 a 255
- Intervallo per un short int	da -32767 a +32767
- Intervallo unsigned short int	da 0 a 65535
- Intervallo per un int	da -32767 a +32767
- Intervallo per un unsigned int	da 0 a 65535
- Intervallo per un long int	da -2147483647 a +2147483647
- Intervallo unsigned long int	da 0 a 4294967295
- Precisione per un float	6 cifre
- Precisione per un double	10 cifre
- Precisione per un long double	10 cifre

Lo *Standard* specifica anche che questi limiti dovrebbero essere presenti come macro per il preprocessore nel file <limits.h>

Data object

Sono state apportate delle variazioni sopratutto per quanto riguarda il collegamento e l'inizializzazione delle variabili (oggetti).

Inizializzazione: Gli oggetti che sono dichiarati 'static' oppure 'auto' possono essere inizializzati facendo seguire alla loro dichiarazione un '=' e una espressione che li inizializza. Oggetti di tipo 'extern' (inter-modulo) saranno considerati più avanti nel paragrafo 'Collegamento'.

Se non si inizializza un oggetto di tipo 'static', a tutti gli elementi di tipo aritmetico presenti nell'oggetto verrà assegnato il valore 0 e tutti i puntatori verranno forzati al valore NULL. Se non si inizializza un oggetto di tipo 'auto', il valore iniziale non sarà definito.

Tutte le espressioni usate per inizializzare oggetti di tipo 'static', vettori, unioni, o strutture devono essere costanti.

Le unioni possono essere inizializzate; il valore di inizializzazione è assegnato al primo elemento dell'unione.

Le espressioni per inizializzare un oggetto scalare (intero, virgola mobile o puntatore) possono essere eventualmente racchiuse fra parentesi. Le parentesi devono invece racchiudere tutte le espressioni usate per inizializzare vettori, strutture e unioni. In una lista di inizializzazione possono esserci più valori che oggetti da inizializzare. Un vettore oppure un puntatore a 'char' possono essere inizializzati con una stringa costante.

Collegamento: (Nota: in questo paragrafo "oggetto" si riferisce a un oggetto dichiarato al di fuori di qualsiasi funzione)

Il collegamento di un oggetto determina la sua visibilità all'interno del programma. Un oggetto con collegamento di tipo esterno è conosciuto da tutti i moduli presenti nel programma. Un oggetto con un collegamento di tipo interno è conosciuto solamente dal modulo in cui è definito. Gli attuali compilatori C spesso discriminano tra i due tipi in modo incompatibile, mentre il problema è stato risolto dallo *Standard*.

Si dice che un oggetto è definito se include un valore di inizializzazione. Un oggetto definito ha un collegamento interno se si specifica la classe di memorizzazione 'static'; altrimenti ha collegamento esterno. Un oggetto può essere definito una volta sola.

Se non è presente un valore di inizializzazione, qualsiasi dichiarazione di oggetto senza il modificatore 'extern' costituisce ciò che è noto come un tentativo di definizione di un oggetto. Se si incontra la definizione di un oggetto nello stesso modulo, tutti i tentativi di definizione vengono considerati come semplici dichiarazioni che si riferiscono a tale oggetto. Altrimenti il primo tentativo di definizione è considerato come una definizione effettiva con valore di inizializzazione uguale a 0.

Il modo più semplice di dichiarare un oggetto con collegamento di tipo esterno è di definirlo in un modulo (con o senza inizializzazione) e referenziarlo in tutti gli altri usando una dichiarazione 'extern'.

Dichiarazioni di funzione

Forse le aggiunte più utili alla definizione del linguaggio sono state effettuate nell'area delle dichiarazioni di funzione, defini-

zioni di funzione e liste variabili di parametri.

Definizioni di funzione: lo *Standard* permette ora di specificare i tipi dei parametri formali all'interno della dichiarazione di una funzione nella parte iniziale della definizione della funzione stessa. Questo nuovo stile di definizione ricorda molto linguaggi come Pascal e Modula-2:

```
main( int argc,char *argv[] )
 {
        /*....*/
 }
```

Se si usa questo stile, deve essere specificato separatamente un tipo per ciascun parametro formale presente nella lista di argomenti. Non è permesso mischiare il nuovo stile con quello definito nel K&R.

Si possono anche definire le funzioni in modo che non abbiano esplicitamente alcun valore di ritorno. Questo tipo di funzioni è detto funzione 'void', definite usando un tipo 'void':

```
void func(int a)
 {
      /*....*/
 }
```

Queste funzioni possono non ritornare alcuna espressione. Se non si specifica esplicitamente un tipo, il default è 'int' per mantenere la compatibilità con il K&R.

Dichiarazioni di funzioni e prototipi: per mantenere lo stile delle definizioni di funzioni, le dichiarazioni di tipo delle funzioni possono includere anche una lista di paramentri formali. Questi parametri consistono in dichiarazioni di tipo con o senza identificatori. L'uso degli identificatori è un puro fatto estetico, e termina il suo scopo alla fine della dichiarazione. Esempi:

```
int main( int, char *[] );
extern char *strcpy( char *dest, const char *src );
```

Bisogna essere consistenti nel fare le dichiarazioni. Ciascuna dichiarazione di funzione deve essere congruente con tutte le dichiarazioni precedenti sia come numero che come tipo di parametri. Le seguenti dichiarazioni evidenziano due casi speciali:

```
extern int getchar( void );
extern int printf();
```

Il primo esempio dichiara esplicitamente che la funzione getchar() non vuole nessun parametro, cioè la lista dei parametri è vuota o 'void'. La seconda dichiarazione asserisce che non è nota alcuna informazione circa il numero e il tipo dei parametri formali. Ciò è stato chiaramente implementato per mantenere la compatibilità con il K&R.

Una dichiarazione di funzione che non specifica il numero e il tipo dei parametri è detta 'prototipo di funzione'. L'aggiunta dei prototipi al C consente al compilatore di effettuare un controllo più stretto sui tipi dei parametri passati a una funzione. Quando si è dichiarato un prototipo per una funzione, ciascuna chiamata successiva a quella funzione è controllata in modo da essere sicuri che si sia passato il numero corretto di parametri. Inoltre, il tipo di ciascun argomento è confrontato con quello che era stato dichiarato nel prototipo. Se risultano differenti, l'argomento viene convertito nel tipo richiesto dal prototipo come se fosse stato assegnato a un oggetto di quel tipo. Non si compiono le promozioni degli argomenti di default ('char' a 'int', 'float' a 'double' ecc.) quando è visibile un prototipo.

Quando, nello stesso modulo, sono presenti sia il prototipo di una funzione che la sua definizione, devono essere entrambi congruenti se la definizione è formulata secondo il nuovo stile. Nelle definizioni che seguono lo stile del K&R, i parametri formali sono prima espansi secondo le promozioni degli argomenti di default e successivamente confrontati con i prototipi.

Se non è presente alcun prototipo nel modulo, la definizione di funzione stessa serve come prototipo per il codice che la segue.

Liste variabili di parametri: alcune funzioni C sono state progettate in modo da accettare un numero variabile di parametri. Sfortunatamente, compilatori diversi usano schemi differenti per gestire queste situazioni, e ciò che funziona in una certa implementazione può non funzionare in un altra. Lo *Standard* fornisce quindi la possibilità di dichiarare esplicitamente questo tipo di funzioni, e rende disponibili delle facilitazioni portabili per gestirle. Una funzione che accetta un numero variabile di parametri è definita terminando la lista dei parametri (solo con il nuovo stile) con una sospensione:

```
int printf( const char *format, ... )
 {
       /* .... */
 }
```

Così, l'unica cosa nota della printf() è che accetta come minimo un parametro, il cui tipo è puntatore a un 'const char'. I prototipi possono anche essere dichiarati nel seguente modo:

```
extern int sprintf( char *dest, const char *format, ... );
```

Il compilatore controllerà che qualunque chiamata a sprintf() abbia almeno due argomenti, e che entrambi siano dei puntatori a 'char'.

Gli argomenti stessi sono accessibili tramite l'uso di speciali macro definite nel file di intestazione <stdarg.h>, che non è descritto in questo articolo.

Il preprocessore

Il preprocessore C, da lungo tempo considerato come una parte integrante del linguaggio, ha beneficiato di una serie di aggiunte e chiarificazioni.

Nuove direttive: è stata aggiunta la direttiva '#elif' come abbreviazione della sequenza '#else #if'.

L'identificatore speciale 'defined' è riservato durante un '#if' o un '#elif' così che:

```
#if defined( NULL )
#if !defined( TRUE )
```

equivalgono a:

```
#ifdef NULL
#ifndef TRUE
```

Altre due nuove direttive sono '#error' e '#pragma'. La prima produce un messaggio di errore in fase di compilazione; la seconda è definita, come uso e come effetti, in funzione dell'implementazione.

Inclusione di file: accanto alle due forme già disponibili:

```
#include <fname1>
#include "fname2"
```

si è aggiunta una terza forma

```
#include fname3
```

dove fname3 è una macro che si espande in una delle altre due forme.

Macro predefinite: nello *Standard* vengono predefinite 5 macro, ognuna delle quali viene espansa ai suoi valori appropriati durante la compilazione del file.

LINE è il numero corrente di linea nel file sorgente.
FILE è il nome corrente del file.
TIME è il tempo corrente di compilazione.
DATE è la data di compilazione.
STDC è la costante decimale 1.
La sua definizione indica un compilatore conforme allo Standard ANSI.

Macro operatori: sono stati aggiunti due nuovi operatori che possono essere usati all'interno di una macro stringa.

L'operatore ## concatena due token del preprocessore adiacenti (un token del preprocessore è una qualsiasi serie consecutiva di caratteri che non siano dei blank). L'operatore # trasforma il parametro che lo segue in una stringa. Per esempio, consideriamo queste definizioni:

```
#define str(s)   # s
#define cat(a,b) a ## b
```

Le seguenti chiamate di macro:

```
str( printf("hello") )
cat( x,2 )
```

verranno espanse in:

```
"printf(\"hello\")"
x2
```

Notate come ciascuna virgoletta presente nel parametro passato a str sia stata modificata in modo da non intaccare la nuova stringa.

La libreria C

Una libreria standardizzata di routine è una risorsa preziosa per i programmatori e inoltre migliora la portabilità. Lo *Standard* definisce una libreria di questo tipo, la quale però è troppo vasta per essere descritta in dettaglio in questo articolo. La Libreria Standard è basata su una libreria compilata da /usr/group, un gruppo di utenti UNIX, ed è stata depurata da tutte le dipendenze da UNIX.

La *Libreria Standard* fornisce inoltre un insieme di intestazioni di libreria. Queste intestazioni contengono i prototipi di funzione per l'insieme di routine che costituiscono la libreria e definiscono delle macro usate comunemente. D' altra parte, le funzioni stesse sono state modificate in modo da risultare invarianti rispetto alla promozioni di default; ciò significa che i parametri passati alla funzione saranno sempre dello stesso tipo, sia che il prototipo sia visibile o no.
Le macro possono anche essere definite in un file di intestazione in modo da prendere il posto delle chiamate alle routine di libreria. Comunque, le routine di libreria devono esistere, poiché le macro possono essere soggette a una direttiva '#undef' da parte dell'utente in qualsiasi momento.

Tra le aggiunte più importanti fatte alla libreria ci sono la gestione degli argomenti variabili, l'informazione sui limiti numerici, e l'informazione locale (l'ambiente corrente).

Le librerie di funzioni del K&R sono state convertite al nuovo stile e alla nuova sintassi, quindi, ad esempio, la 'malloc' ora ritorna un 'void *' invece che un 'char *'.

Miscellanea

Al linguaggio sono stati apportati altri numerosi cambiamenti e aggiunte. Qui di seguito c'è un breve sommario dei punti più importanti:

- Sono stati aggiunti i caratteri '\a' e '\v' per l'allarme (bell) e la tabulazione verticale.

- Sono state aggiunte delle serie di speciali caratteri trigraph come equivalenti dei caratteri ASCII che non possono comparire negli insiemi di caratteri di altri paesi; un trigraph è una sequenza di tre caratteri che iniziano con ??

- I suffissi u e l possono essere usati con le costanti intere in modo

da specificare dei valori 'unsigned' e 'long'; entrambi possono essere usati insieme per specificare un 'unsigned long'.

- I suffissi f e l possono essere usati con delle costanti in virgola mobile in modo da specificarne i valori 'float' e 'long double'.

- Se il bit più alto di una costante ottale o esadecimale è posto a 1, la si considera 'unsigned'.

- Le stringhe adiacenti separate solamente da uno spazio vengono concatenate.

- è stato aggiunto il più unario ('+') in modo da forzare che la valutazione di una espressione aritmetica avvenga prima di ogni altra.

- Una funzione può essere chiamata attraverso un puntatore usando o la sintassi secondo lo stile del K&R (*fp)() oppure secondo il nuovo stile fp().

- La lunghezza significativa di un identificatore esterno è stata mantenuta a 6 caratteri senza case sensitivity, per mantenere la compatibilità con i linker esistenti.

- La lunghezza significativa degli identificatori interni è come minimo di 31 caratteri (con case sensitivity). □

Alcuni esempi di definizioni di pointer

XPointer: 'pointer a tre colori
```
DATA 9,9,-5,-4
DATA &H0000,&H0000
DATA &HC180,&H4100
DATA &H6380,&HA280
DATA &H3700,&H5500
DATA &H1600,&H2200
DATA &H0000,&H0000
DATA &H1600,&H2200
DATA &H2300,&H5500
DATA &H4180,&HA280
DATA &H8080,&H4100
DATA &H0000,&H0000
```
XPlain: 'a un solo colore (colore 01)
```
DATA 9,9,-5,-4
DATA &H0000,&H0000
DATA &H8080,&H0000
DATA &H4100,&H0000
DATA &H2200,&H0000
DATA &H1400,&H0000
DATA &H0000,&H0000
DATA &H1400,&H0000
DATA &H2200,&H0000
DATA &H4100,&H0000
DATA &H8080,&H0000
DATA &H0000,&H0000
```
SPointer: 'tre colori separati
```
DATA 9,9,-5,-4
DATA &H0000,&H0000
DATA &H0FC3,&H0000
DATA &H3FF3,&H0000
DATA &H30C3,&H0000
DATA &H0000,&H3C03
DATA &H0000,&H3FC3
DATA &H0000,&H03C3
DATA &HC033,&HC033
DATA &HFFC0,&HFFC0
DATA &H3F03,&H3F03
DATA &H0000,&H0000
```
BPointer: '"scatola" a due colori (colori 01 e 11)
```
DATA 13,16,-8,-6
DATA &H0000,&H0000
DATA &HFFFE,&HFFFE
DATA &HC106,&HC006
DATA &HC106,&HC006
DATA &HC106,&HC006
DATA &HC106,&HC006
DATA &HC106,&HC006
DATA &HFFFE,&HC006
DATA &HC106,&HC006
DATA &HC106,&HC006
DATA &HC106,&HC006
DATA &HC106,&HC006
DATA &HC106,&HC006
DATA &HFFFE,&HFFFE
DATA &H0000,&H0000
```

(segue da pagina 59)

Come cambiare il "mouse pointer" in AmigaBasic

```
'** Usa il nuovo pointer **

CALL SetPointer(WINDOW(7),mem&,pHeight%,pWidth%,
pXOffset%,pYOffset%)
PRINT "SetPointer"
PRINT "(Premere il tasto sinistro del mouse per ripristinare il
pointer originario)"

'** Attendi che l'utente prema il tasto sinistro del mouse **

WHILE MOUSE(0) <> -1
WEND

'** Ripristina il pointer di sistema **

CALL ClearPointer(WINDOW(7))
PRINT "ClearPointer"

'** Rilascia la memoria usata e chiudi le library **

CALL FreeMem(mem&,pSize&)
LIBRARY CLOSE
END
```

Il Manx C 3.6 e l'SDB

di Nick Sullivan

Potrebbe migliorare, ma va già bene.

Manx Aztec 3.6a C Development System (Versione per developer): $299 Upgrade dalla versione 3.4: $30 (o gratis con SDB) SDB Source Level Debugger: $75

La versione 3.6 del Manx Aztec C non costituisce per vari aspetti un miglioramento sostanziale rispetto alla release precedente, la versione 3.4. Infatti, le prestazioni dei programmi principali (compilatore, assemblatore e linker) sembrano più o meno le stesse, sia relativamente alla velocità di esecuzione che per quanto riguarda le dimensioni e la velocità del codice generato. La Manx ha lavorato per rendere il compilatore conforme allo standard ANSI e su miglioramenti della struttura stessa del compilatore, che dovrebbero permettere una migliore generazione del codice, ma i cambiamenti non sono così radicali.

Ciò che è nuovo nel 3.6 è la possibilità di usare l'SDB, il debugger a livello di sorgente da lungo tempo atteso. Come vedrete in seguito, l'SDB ha bisogno di avere molte informazioni sui file di sorgente relativi al programma che si sta correggendo. Queste informazioni vengono ottenute da un file speciale ".dbg" che viene generato dal linker (tramite una opzione specifica). Il linker, dal canto suo, può estrarre le informazioni attinenti al debugging dai file oggetto (".o"), a patto che anche il compilatore sia stato chiamato con un'opportuna opzione. L'implementazione di queste opzioni è la sola innovazione significativa della release 3.6.

Gli altri cambiamenti sono migliorie secondarie. Alcuni bug (errori) della versione 3.4 sono stati corretti e sono state aggiunte nuove opzioni. La più interessante di queste è la possibilità di generare codice in-line per le funzioni strcpy(),strcmp() e strlen(), sia per tutte quante (per mezzo di un'opzione) sia individualmente, facendo precedere il nome della funzione dalla stringa "_BUILTIN_".

Come ho già precisato, però, questa versione del compilatore esiste principalmente per rendere possibile l'uso di SDB e questo prodotto è di interesse più che notevole.

Perché un debugger a livello di sorgente?

Chiunque abbia usato un debugger convenzionale a livello di linguaggio macchina, come il DB della Manx o il Metascope prodotto dalla Metadigm, per correggere programmi scritti in C si sarà reso conto della necessità di un programma specializzato per il debugging a livello di sorgente. I debugger convenzionali hanno accesso solo a quei simboli che sono globali nel sorgente del programma; in C questo significa che un debugger di questo tipo conosce solo i nomi delle variabili e delle funzioni dichiarate come esterne e può riferirsi ad altre solo con indirizzamento assoluto, con spiazzamento rispetto a un registro o a una variabile globale oppure tramite un nome di registro. Di conseguenza il programmatore spreca molto tempo a calcolare indirizzi quando intende esaminare il contenuto di una variabile "static", automatica o di tipo "register" oppure disassemblando una funzione "static".

Ci sono anche altri motivi di frustrazione. Un debugger convenzionale presenta riferimenti a una struttura di un programma C come spiazzamenti dall'indirizzo base della struttura e non col nome del membro appropriato. Supponiamo che A0 contenga un puntatore a una struttura di tipo Window e che si trovi l'istruzione "move.l 52(a0),d0"; svelti, a quale membro della struttura ci riferiamo? Se avete detto "IDCMPFlags" probabilmente non avete bisogno di un "source level debugger": ciò di cui avete veramente bisogno sono delle relazioni sociali. Inoltre, eseguire passo-passo a livello di linguaggio macchina un programma C può sì risultare molto illuminante, ma di scarsa efficienza per un rapido debugging.

Il debugging a livello di sorgente fornisce una risposta a tutti questi problemi. Se volete conoscere il valore di una variabile "auto", potete riferirvi a essa col suo nome. Se volete esaminare quella data struttura Window, potete leggere sullo schermo l'intera lista dei suoi membri, con i nomi, i tipi e i valori che hanno in questo momento. Potete anche eseguire passo-passo secondo le istruzioni del programma sorgente e dimenticarvi del tutto del linguaggio macchina.

Breakpoint

Eseguire passo-passo equivale a porre un breakpoint nel programma dopo ogni istruzione. Questa non è una procedura molto flessibile. Generalmente si inizia con una vaga idea dell'area del codice in cui si genera l'errore e ci si vuole arrivare rapidamente per poi eseguirla passo-passo. Spesso ci si vuole fermare solo in uno o pochi punti del programma ed esaminare il contenuto delle variabili o di altre zone di memoria (ad esempio, entrando o uscendo da una funzione sospetta). In queste situazioni, dovete poter porre breakpoint in punti qualsiasi del codice e far girare il programma a tutta velocità finché non si capita su uno di essi.

SDB offre alcuni tipi di breakpoint, nei quali variano le condizioni sotto cui il breakpoint viene preso in considerazione. Un

breakpoint semplice arresta l'esecuzione e restituisce il controllo quando viene raggiunta la locazione desiderata; si può specificare l'indirizzo di tale locazione come indirizzo fisico, nome di funzione, linea del sorgente, in uno qualsiasi dei file da cui il vostro programma è composto, o espressione C che fornisca un indirizzo. Si può specificare opzionalmente uno 'skip count', cioè il numero di volte che il breakpoint deve essere raggiunto dal controllo prima che l'esecuzione venga fermata. Potete poi fornire anche una serie di comandi SDB (ad esempio, per mostrare il contenuto di una variabile fondamentale) che vanno eseguiti automaticamente ogni qualvolta venga effettuato quel certo breakpoint. Un'altra forma è costituita dall'"expression" breakpoint, che permette di specificare un'espressione C arbitraria da valutarsi all'entrata o all'uscita di ogni funzione (o dopo ogni istruzione in fase di esecuzione passo-passo). L'esecuzione viene fermata quando tale espressione assume un valore diverso da zero.

Una questione di interpretazione

La capacità di comprendere espressioni C inserite al momento del debugging è estremamente utile. A parte il suo uso con i breakpoint e con alcuni altri comandi, questa possibilità può essere impiegata direttamente per valutare un'espressione e ottenerne il risultato. Poiché le chiamate di funzione e gli assegnamenti sono espressioni (o sotto-espressioni) C consentite, potete chiamare manualmente ogni funzione nota al vostro programma con svariati valori di prova e determinare o modificare il contenuto delle variabili. Questo non è esattamente C interpretato, ma senz'altro fornisce molte delle comodità che un interprete C vi darebbe.

Per mostrare in maniera chiara le variabili, viene fornito un comando di stampa formattata. Ecco un esempio del suo uso:

```
>p *window->RPort->BitMap
struct BitMap = {
unsigned int BytesPerRow = 80;
unsigned int Rows = 200;
unsigned char Flags = 0;
unsigned char Depth = 2;
unsigned int pad = 0;
unsigned char *Planes[0] = 0x00002e68
}
```

Per determinare rapidamente se una struttura del vostro programma è stata inizializzata nel modo corretto, questo è uno dei metodi più comodi che si possa trovare.

SDB e lo stack

Il C impiega lo stack per tre scopi principali: come area di memoria per le variabili automatiche, per passare gli argomenti delle funzioni e per salvare gli indirizzi di ritorno per le chiamate di funzione. Dal momento che la locazione di tutti questi oggetti è nota o può essere dedotta dalla conoscenza di come il compilatore usa lo stack, l'SDB è in grado di fornire uno "stack backtrace" da un punto qualunque del programma, mostrando le chiamate di funzione annidate e i loro parametri fino a giungere a _main() e può mostrare non solo le variabili "auto" della funzione corrente, ma anche di tutte le funzioni parzialmente eseguite per arrivare a quella corrente. Questa informazione è spesso di aiuto per rendersi conto di che cosa è accaduto in situazioni complesse - per esempio, quando si lavora con funzioni che chiamano se stesse ricorsivamente.

Figlio di DB

Gli utenti del debugger a livello di linguaggio macchina della Manx, DB, noteranno una più che casuale somiglianza tra tale programma e SDB, somiglianza che si estende fino alla sintassi dei comandi. Infatti, SDB incorpora molte delle funzionalità offerte da DB, fornendo potenti mezzi per il debugging a livello sia di linguaggio macchina sia di sorgente. Per esempio, l'utente può liberamente passare dall'esecuzione passo-passo a livello di sorgente a quella a livello di linguaggio macchina. I comandi per disassemblare la memoria, per fare un "dump" della memoria e dei registri o modificarli, per stampare informazioni specifiche di Amiga come le liste dei task o dei device, per creare macro e per ridirigere l'output del debugger sono stati presi direttamente da DB. Molti dei comandi relativi ai breakpoint sono identici o quasi ai loro corrispettivi sia nella sintassi sia nel funzionamento e perfino lo "stack backtrace" e i comandi per la stampa formattata si appoggiano ai precedenti stabiliti da DB.

La differenze sono interessanti quanto le somiglianze. E' molto ovvio che SDB sia più vicino all'interfaccia utente di Amiga, anche se ciò avviene in una maniera piuttosto strana. Il programma gira in una finestra suddivisa in tre aree: un'area nella parte superiore per mostrare il sorgente; uno "string gadget" per l'inserimento dei programmi (non uno di Intuition, ma uno creato appositamente) che occupa l'intera riga sottostante e un'area inferiore dedicata all'output. Le dimensioni relative dell'area del sorgente e dell'output possono essere aggiustate facendo scorrere su o giù lo "string gadget", sia dalla tastiera che con dei gadget. La finestra stessa è ridimensionabile e i colori di ogni area sono separatamente selezionabili. Questa è una disposizione poco convenzionale (rispetto, ad esempio, a quella a finestre multiple del Metascope), ma presenta il vantaggio di poter togliersi di torno il debugger mandando una sola finestra dietro le altre eventualmente presenti.

Nonostante la finestra, i gadget e i colori, SDB ha ancora molta strada da fare nel settore dell'interfaccia utente. Tutti i comandi vanno impartiti mediante codici (più o meno) mnemonici: 'bs' per porre un breakpoint, 'acs' per selezionare i colori dell'area del sorgente e così via.
Ci sono più di sessanta codici, molti dei quali accettano diversi tipi di argomenti; potreste impiegare molto tempo prima di poter fare a meno del manuale. Considerata la flessibilità di Amiga nell'input da utente, non dovrebbe risultare difficile per la Manx rendere le future release del programma più facili da imparare e da controllare, e spero che lo facciano. Per esempio, sarebbe comodo se si potesse selezionare con il mouse una linea nell'area del sorgente per metterci un breakpoint.

(segue a pagina 75)

ARexx

di Harry Phillips

Ecco il linguaggio REXX per Amiga

"Taglia a fette e a quadratini! Affetta le dita o i pomodori maturi con la stessa facilità! Ma, attenzione! E' anche una cera per pavimenti e un concime per il giardino!"

Sono sicuro che abbiate tutti ascoltato e odiato il conduttore di vendite televisive che presenta l'ultimo ritrovato che fà così tante cose e che costa così poco. A me è capitato e qualche volta mi domando se gli altri non mi considerino alla stessa stregua quando qualcuno mi chiede a che cosa serve ARexx. Come vedrete, ARexx è un vero attrezzo del mestiere per il quale è difficile esagerare nel tesserne le lodi.

Ci sono diversi aspetti di ARexx che risaltano subito. Come linguaggio per file di comandi (script) è simile, benché molto più potente, al linguaggio dell'AmigaDOS impiegato nei file eseguiti dal comando EXECUTE.

ARexx consente uno stupefacente controllo dei comandi di sistema, cominciando dal potente costrutto DO, che permette i cicli controllati, fino alla capacità di fare il "parsing" di una forma qualsiasi di input, sia essa un file e il suo contenuto, una lista di directory o un input da tastiera.

Aggiungete ora a ciò la possibilità della comunicazione tra processi, che permette a un programma ARexx di interagire sia con un altro programma ARexx sia con qualsiasi applicazione dotata di un'interfaccia ARexx e potete immaginare perché io sia ora tentato di chiedervi "Adesso, quanto vi aspettereste di pagare? Ma ATTENZIONE! C'è dell'altro!".

ARexx viene fornito con due library, completamente documentate, che possono essere usate con qualsiasi programma. Le librerie forniscono una base d'appoggio tanto per l'impiego delle funzioni di ARexx quanto per funzioni di alto livello più generali. Infine, se non ho dimenticato qualcosa, ARexx consente l'uso di librerie esterne in maniera tale che possiate estendere voi stessi il linguaggio.

Diamo per prima cosa un'occhiata alle possibilità offerte a livello di script rispetto a quelle fornite dall'AmigaDOS. Ecco una lista di funzionalità comprese in ARexx che possono essere realizzate con uno script dell'AmigaDOS solo con molta difficoltà, sempre che ci si riesca.

Cicli: I cicli sono ottenuti in due maniere, sia saltando a un'etichetta sia mediante costrutti specifici. Questi ultimi sono estremamente flessibili, consentendo un numero specificato di iterazioni, l'uscita condizionale e perfino nessuna uscita del tutto.

Stringhe: ARexx rende possibile la manipolazione diretta delle stringhe. Possono essere definite in un programma o possono essere inserite dall'utente attraverso la tastiera, da un file o come risultato di un comando. Le stringhe possono essere divise, concatenate, scandite o manipolate come pure stringhe o come liste di parole, in quasi ogni maniera immaginabile.

Selezione: ARexx fornisce una parola riservata SELECT per il controllo del flusso di programma. Assomiglia molto a una serie di costrutti IF-THEN e ha un caso di default utilizzato quando nessuno dei casi specificati è verificato. Diversamente da costrutti equivalenti in altri linguaggi, SELECT opera la selezione valutando una qualsiasi espressione condizionale.

Calcoli: Possono essere effettuati calcoli su valori interi, a virgola mobile o booleani. ARexx è un linguaggio privo di tipi, nel senso che potete effettuare un'operazione tipica delle stringhe o matematica sulla stessa variabile; il successo di tale operazione dipende unicamente dal fatto che l'oggetto su cui applicate l'operazione sia adatto o meno.

Procedure: Le procedure sono implementate in modo molto simile a quello con cui appaiono in altri linguaggi. Sono sottoprogrammi con uno spazio privato per lo stack e per le variabili. Le variabili possono essere lasciate nascoste nella procedura oppure possono essere "esposte" in modo che una procedura possa modificare od ottenere valori da variabili esterne a essa.

Comunicazione: Un programma ARexx può comunicare con altri programmi ARexx o con un programma opportunamente scritto in un altro linguaggio qualsiasi. Questa possibilità potrebbe risultare la caratteristica più importante di tutto ARexx. Vedremo in seguito la ragione di ciò.

Tracing: ARexx fornisce funzioni molto potenti per il tracing, solitamente limitato alla sola esecuzione passo-passo. Il tracing può essere attivato o disattivato all'interno di un programma ARexx e può essere effettuato su comandi, errori, risultati intermedi, etichette, risultati finali o su tutti insieme. Esiste un modo "scan", per mezzo del quale si attraversano tutte le istruzioni senza che peraltro venga eseguito nulla, e un modo di "trace interattiva" che è molto comodo per eseguire passo-passo un programma o per provare un'istruzione ARexx direttamente da tastiera.

Interrupt: ARexx permette di associare azioni alla ricezione di diversi interrupt, come i segnali relativi a CTRL-C, CTRL-D, CTRL-E o CTRL-F, e a errori generici, all'impiego di variabili non inizializzate e così via.

Clipboard: ARexx supporta interamente una clipboard; non è la clipboard standard di Amiga, ma è comunque molto utile. Questa clipboard può essere usata per passare stringhe tra programmi e perfino per ospitare un programma ARexx da far eseguire a un altro programma ARexx.

Interprete: Questa è davvero una delle più esoteriche tra le caratteristiche di ARexx. Rende possibile l'esecuzione da parte di ARexx di una stringa come se fosse un programma. Riferitevi a "calc.rexx", un programma molto corto ma di grande utilità, poco più avanti per un esempio pratico del suo uso.

A questo punto potrebbe essere interessante dare un'occhiata a qualche semplice script ARexx. Dovrei osservare che io carico la "rexxsupport.library" durante la startup-sequence per evitare di doverne controllare la presenza ed eventualmente di doverla caricare in ogni programma che ne faccia uso. Uso anche WShell, un altro prodotto eccellente di William Hawes che mi permette di chiamare programmi ARexx senza dover digitare "rx". Se, infatti, non usate WShell dovrete usare RX nella stessa maniera con cui gli script dell'AmigaDOS richiedono il comando EXECUTE, almeno sotto l'AmigaDOS 1.2.

Il primo di questi script è incluso nel package di ARexx e mostra l'impiego dell'istruzione INTERPRET. Viene invocato con il comando "calc <expression>", dove <expression> è una qualsiasi espressione valutabile da ARexx. Questo comando funge da piccola calcolatrice e devo dire che viene usato molto spesso. Ecco in che cosa consiste il comando:

```
/* calc.rexx */
interpret 'say' arg(1)
```

La prima cosa in un qualsiasi programma ARexx è un commento. ARexx usa questo commento per verificare che si tratti di un programma vero e proprio. Il commento e il corpo del programma avrebbero potuto essere scritti su una sola linea. La seconda linea è quella che svolge tutto il lavoro; costruisce una stringa costituita dalla parola "say" seguita da uno spazio e dagli argomenti forniti al programma come <expression>. Il comando INTERPRET esegue quindi questa stringa come se fosse un'istruzione di un programma ARexx. Supponendo che io abbia chiamato questo programma battendo "calc 6*5", la stringa sarebbe "say 6*5" (un'istruzione corretta in ARexx, essendo "say" il comando che stampa sul video) e il risultato sarebbe "30" stampato sullo schermo.

Ho affermato che <expression> può essere una qualsiasi espressione ARexx corretta, il che ci consente di fare alcune cose con il nostro primo esempio che potrebbero non risultare evidenti a prima vista. Ad esempio, potrei scrivere "calc c2d('a5'x)". Questo mi farebbe apparire "165" sullo schermo, essendo 165 il risultato della conversione a decimale del numero esadecimale A5. Un altro interessante (ma non troppo utile) esempio potrebbe essere "calc words('Del tutto inutile')", che darebbe come risultato "3", il numero di parole nella stringa "Del tutto inutile".

Forse è il caso di vedere qualcosa di più interessante. Eccovi allora un piccolo script che vi consente di usare un "*" come "wildcard" per il comando AmigaDOS COPY:

```
/* copy — aggiunge la wildcard * al comando copy */
parse arg x
asterisk = pos('*',x)
do while asterisk > 0
   x = delstr(x,asterisk,1)
   x = insert('#?',x,asterisk-1,2)
   asterisk = pos('*',x)
end
c:copy x
```

Ancora una volta, abbiamo un commento ad aprire il programma. "parse arg x" trasferisce tutti gli argomenti specificati sulla linea di comando alla variabile "x". Si noti che non dobbiamo dichiarare x come variabile e non dobbiamo neppure dire di che tipo di variabile si tratti. La linea "asterisk = pos('*',x) assegna alla variabile "asterisk" un numero che corrisponde alla posizione nella stringa x del primo "*" che essa contiene. Se x non contiene alcun "*", la variabile asterisk assumerà il valore 0.

Il ciclo "do" sarà eseguito se il valore di "asterisk" è diverso da 0 e continuerà a essere eseguito fintanto che ci saranno "*" in x. La prima occorrenza del carattere "*" viene rimossa eseguendo la linea "x = delstr(x,asterisk,1)" e sostituita con la wildcard standard di AmigaDos "#?" nella linea successiva. Si continua a controllare la presenza di altri asterischi, sostituendoli quando li troviamo, e quando sono terminati, chiamiamo c:copy con gli argomenti modificati contenuti nella variabile x.

Programmi che conversano

Sebbene sia facile entusiasmarsi per le caratteristiche di ARexx in qualità di linguaggio per script e sia facile lasciar correre a briglie sciolte la vostra immaginazione sui possibili programmi che vi renderebbero la vita facile, la qualità più sorprendente di ARexx sta nella possibilità di comunicare con altri processi. I programmi ARexx possono comunicare uno coll'altro, il che significa che potete far partire programmi ARexx che aprono un port con un certo nome e attendono un certo messaggio. Quando viene inviato un messaggio a un port da un altro programma, il programma che ha aperto quel port può svegliarsi, analizzare il messaggio ed effettuare tutte le operazioni che il programmatore desidera.

Ogni programma, indipendentemente dal linguaggio in cui è scritto, può comprendere i messaggi ARexx. Dotare un programma di un'interfaccia ARexx è una cosa piuttosto semplice e permette un grado di flessibilità notevole. Attualmente ci sono due editor, mentre un altro dovrebbe aggiungersi tra breve, che possono servirsi di messaggi ARexx: TxEd Plus e Uedit.

Per apprezzare appieno ciò che si può fare con un programma come TxEd Plus connesso ad ARexx, considerate che è possibile prendere il controllo di TxEd Plus per fargli portare a termine tutte le operazioni che normalmente si possono fare attraverso la tastiera, ricevendo eventualmente anche dei dati. Questi dati

possono essere il contenuto della linea corrente, il titolo del file che si stava modificando e molte altre cose ancora.
Se avete mai ammirato l'interfaccia utente del Turbo Pascal, che consente di scrivere un programma, compilarlo, farne il linking e il debugging, senza mai abbandonare l'editor, sappiate che adesso tutto ciò può essere fatto in maniera molto semplice con ARexx, TxEd Plus e il vostro compilatore preferito. Un programma ARexx controlla l'intero processo, attendendo uno specifico messaggio, che voi potete inviargli tramite un menu o una sequenza di tasti, per far compilare il programma che avete nell'editor. Il programma ARexx può chiedere a TxEd Plus di salvare il programma, chiamare il compilatore e notificare il successo o il fallimento della compilazione. Se ci sono stati errori, può leggere il file degli errori, analizzarlo per comunicarvi il tipo degli errori e posizionare il cursore su ognuna delle linee in cui si è generato un errore. Se la compilazione è invece avvenuta senza errori, potreste, con un'altra selezione di un menu, provare il programma. In questa maniera, a differenza del Turbo Pascal, potete scegliere liberamente l'editor e il compilatore che preferite.
Come vi sarete accorti, l'editor diventa più di un programma per elaborare testi. Diventa tutto quel che volete: un lettore intelligente di file o uno strumento per la lettura dei messaggi su un BBS, corredato eventualmente della possibilità di rispondere a essi, scartarne alcuni e tenere in un archivio altri. Provate a immaginare qualche impiego e vi renderete subito conto del perché io sia così entusiasta di ARexx.

Un altro esempio notevole è PCLO Plus, prodotto dalla SoftCircuits. PCLO Plus è un programma per progettare e disegnare circuiti stampati ed è il fratello maggiore di PCLO, uscito diverso tempo fa. L'aggiunta di un'interfaccia ARexx ha reso questo programma, già interessante, un'applicazione che utenti di altre macchine possono solo sognare.
Al momento attuale, un numero crescente di applicazioni vengono dotate di un'interfaccia ARexx. Per ora i programmi disponibili con interfaccia ARexx sono AmigaTEX (un programma tipografico), PCLO Plus, TxEd Plus, Uedit e forse altri di cui non sono al corrente. Microfiche Filer, un sistema d'archiviazione unico nel suo genere, e un database ancora non presentato ufficialmente stanno per uscire corredati di interfaccia ARexx. Girano, inoltre, voci secondo le quali "Access!" un programma shareware di terminale potrebbe venir dotato di un'interfaccia ARexx.
L'aspetto più interessante di tutto questo è che voi, gli utenti finali, potete liberamente determinare quali programmi usare, riunendo i tool che soddisfano i vostri specifici bisogni piuttosto che affidarsi alla capacità di un autore di programmi di prevedere tutti i vostri desideri.

Concluderò questa discussione su ARexx in qualità di mezzo di comunicazione tra processi con alcune considerazioni. Supponete di avere un programma di terminale molto semplice che non disponga di protocolli di trasferimento file, non abbia un linguaggio dedicato per script, né un file requester, né un'agenda telefonica, e così via. Immaginate ora che esso possa disporre, sotto il controllo di un programma ARexx; di menu definibili dall'utente, che una volta selezionati, facciano compiere al programma ARexx determinate operazioni. Tale programma potrebbe poi intercettare certe stringhe definite dall'utente e notificare il loro arrivo all'utente stesso. Sarebbe poi particolarmente interessante se si potesse ordinare al programma di terminare la lettura della porta seriale. Provate adesso a figurarvi ciò che potrebbero fare un file requester, un paio di programmi di protocollo che sappiano come fare l'"uploading" o il "downloading" e un po' di gadget che controllino il tutto via ARexx.
Si può facilmente immaginare come usare tutti questi moduli per creare un emulatore di terminale con un gran numero di possibilità, tenendo il tutto assieme con un programma ARexx, ma si può fare anche di più. Se non c'è il vostro protocollo preferito o se il file requester non è di vostro gradimento, potete sempre scrivervene uno voi come meglio vi aggrada o convincere qualcuno a farlo. Per esempio esiste già un file requester di pubblico dominio con interfaccia ARexx.
Possiamo perfino non accontentarci solo di un programma di terminale. Lo stesso insieme di moduli usato con un altro programma ARexx può diventare un programma di BBS del tutto automatico, un BBS stesso e, con altri moduli, un gioco o una serie di giochi che sfruttino il modem. Che cosa ne direste di un programma di terminale con un'area per il testo e una per la grafica? Un semplice programma grafico potrebbe diventare un programma di terminale grafico piuttosto sofisticato.

Dulcis in fundo, il package di ARexx include una library, in cui sono contenute funzioni di alto livello che potete chiamare dai vostri programmi. Ci sono funzioni per il trattamento delle stringhe, per la manipolazione di file, per conversioni numeriche, invio di messaggi o di pacchetti dos, tracciamento delle risorse impiegate e altro. E' compresa anche una dettagliata documentazione per il programmatore, insieme con istruzioni e consigli per inserire un'interfaccia ARexx nei vostri programmi e per scrivere librerie di funzioni per estendere il linguaggio ARexx stesso; qualcuno infatti ha già scritto una library per l'uso della porta seriale.

ARexx è stato scritto da William Hawes, che è anche l'autore di ConMan, un programma shareware per rendere piu comodo e rapido l'uso della CLI. Ha anche scritto WShell, un programma che si accompagna molt bene ad ARexx.

Io considero ARexx come uno dei migliori programmi che abbia mai acquistato per il mio Amiga; con l'aggiunta di ConMan e di WShell, ARexx costituisce per me un ambiente di lavoro di tutto rispetto facilmente riconfigurabile. Potrei perfino arrivare a sostenere che ARexx sia il programma più importante che sia uscito per Amiga e che avrà senz'altro un impatto rilevante sul modo con cui usiamo questa macchina.

Come potreste già aver intuito a questo punto, raccomando il suo acquisto senza riserva alcuna.

ARexx costa $49.95 ed è disponibile presso l'autore:
William S. Hawes
P.O. Box 308
Maynard, MA 01754
USA
(617)568-8695.

Uno sguardo alla struttura dei dischi di Amiga

di Betty Clay

Per comprendere come sono strutturati i file sul disco su Amiga, è necessario imparare alcuni termini nuovi, piuttosto differenti da quelli a cui sono stati abituati fin dagli albori gli utenti Commodore.
Siamo abituati a una traccia dedicata alla directory, cioè a una "directory track"; su Amiga si parla invece di "hash-chain", catena hash, e di "root block", blocco principale. Parliamo della BAM; su Amiga ci sono pagine bit-map.
Siamo abituati a una sola directory sulla traccia centrale, mentre su Amiga ci possono essere diverse directory in varie posizioni e perfino di tipi differenti. L'Amiga prevede al loro posto i root directory block, gli user-directory block, gli header block (blocchi di testa), anche detti "key", i list block, gli extension block e i data block per non parlar poi dei corrupt block e dei corrupt block cancellati!
Comunque, una volta imparati i nuovi termini, si potrà osservare che i due sistemi sono, per vari aspetti, simili.

Su Amiga la struttura delle directory ha inizio con il root block, che contiene la posizione delle altre directory sul disco e di quei file che non appartengono alle altre directory.
Così come eravamo abituati a trovare la directory sulla traccia centrale, sotto AmigaDOS il root block è situato nella traccia centrale.

Con il sistema operativo attuale, un disco viene formattato secondo ottanta tracce, ognuna avente ventidue settori di 512 byte ciascuno. Ogni settore è diviso in 128 slot (caselle) da 32 bit; ogni slot può essere visto come un singolo valore a 32 bit, come due valori a 16 bit o come quattro byte. Nel gergo di Amiga le tracce sono dette cilindri (cylinder) e i settori sono chiamati blocchi, un termine, questo, già familiare. C'è anche un numero per specificare la superficie del disco da considerare; tale numero è sempre zero secondo il disk editor DiskEd, mentre può essere uno o zero per DiskDoctor!

La directory principale su un disco di Amiga si trova al blocco 880 (traccia 40, settore 0) sotto il presente sistema operativo. Tale blocco ha 128 slot disponibili per contenere informazioni, numerati dallo 0 al 127. Lo slot zero contiene l'identificatore del tipo di file (un '2' per uno "short file").
Gli slot uno e due nel root block contengono sempre zero; lo slot tre contiene ora come ora 72, ma potrebbe contenere altri valori in futuro. Settantadue è, infatti, il valore ottenuto sottraendo dalle 128 caselle disponibili il numero di slot contenenti informazioni per l'identificazione (i primi sei e gli ultimi cinquanta). Se un futuro DOS sarà predisposto per blocchi più lunghi di 512 byte, ci saranno allora più di 72 blocchi liberi per la "hash table" e lo slot tre conterrà un valore diverso.
Il quarto slot non è usato sul root block, mentre il quinto contiene il "checksum"; questo avviene qualsiasi sia il tipo di blocco considerato.

Le hash table

Gli slot dal sesto al settantasettesimo dei directory block contengono la "hash table".
"Hashing" è un processo matematico mediante il quale viene associato al nome di un file un numero per mezzo di una formula, facendo in modo che ogni possibile nome dia luogo a un codice numerico entro un intervallo specifico, nel nostro caso un numero tra 6 e 77, inclusi. La formula è già presente nel DOS e viene automaticamente calcolata, per cui non è affatto indispensabile conoscerla esattamente. Per capire la ragione della hash table, dobbiamo solo aver chiaro che ogni possibile nome di file viene associato a un numero che va da 6 a 77, gli stessi numeri degli slot dedicati alla hash table.

Se il nome di un file ha un valore hash di dodici (ed è il primo file nella directory ad aver quel numero hash), allora il numero del blocco in cui tale file ha inizio viene posto nello slot dodici del directory block.
Ogni slot contiene il numero del blocco di inizio di un file il cui nome ha un valore hash pari al numero dello slot. Se le cose finissero qui, ciò significherebbe che una directory sarebbe limitata ad avere non più di settantadue file e non ci potrebbero essere due file con lo stesso valore hash. Questa sarebbe una limitazione assolutamente intollerabile, per cui risulta necessario introdurre la cosiddetta "hash-chain" (catena hash).

La hash chain

Supponiamo che abbiate salvato un file con il nome "Filename" nella directory principale. "Filename" ha un valore hash pari a 59. Supponiamo che il blocco di inizio di questo file sia il blocco 884. Lo slot 59 della directory principale conterrà dunque il numero 884. Supponiamo che desideriate salvare un altro file nella directory principale e che intendiate chiamarlo "File4", nome che ha anch'esso un valore hash di 59; per fissare le idee, supponiamo che il file inizi al blocco 985.
Quando verrà esaminata la directory principale prima della memorizzazione del "File4", ci si accorgerà che lo slot 59 contiene già il numero 884. Il blocco 884 verrà allora letto e ne verrà esaminato lo slot 124 (quello dedicato alla hash chain) per

vedere se contiene un numero di blocco oppure uno zero. Nel caso in cui contenga un numero di blocco, il blocco corrispondente a quel numero verrebbe esaminato per vedere se il suo slot 124 contiene uno zero e così via fino a quando non si trova nella catena un blocco che abbia uno zero in tale casella.
A questo punto, lo zero verrebbe sostituito da 985 e con tale operazione il file verrebbe aggiunto correttamente nella hash chain. Solo ora il file "File4" potrebbe essere realmente scritto sul disco, con il suo header block nel blocco 985.
Il quarto slot del blocco 985 conterrà il numero del prossimo blocco impiegato per "File4" e lo slot 124 conterrà uno zero, segnalando così che si tratta dell'ultimo file attualmente presente su questa hash chain. Si noti che è proprio la hash chain che consente all'Amiga di avere un numero praticamente illimitato di oggetti in ogni directory, richiedendo poche operazioni per localizzare uno specifico file.

Quando ordinate che il "File4" venga letto, viene seguito un procedimento simile. Il nome viene nuovamente convertito nel suo valore hash, ovvero 59; quindi viene letto lo slot 59 del blocco della directory principale e la presenza di 884 in quella casella comporterà la lettura del blocco 884. A questo punto si confrontano i nomi dei due file e, dato che non sono uguali, verrà letto lo slot 124 per ottenere il numero del blocco di testa del prossimo file nella catena hash.
Nel nostro esempio, la casella 124 conterrà 985 e verrà dunque letto il blocco 985; constatato che i nomi dei due file coincidono, il file potrà essere finalmente caricato.

Gli altri cinquanta slot

Al termine del root block ci sono altri cinquanta slot che contengono informazioni relative al disco. La casella 78 contiene un flag che specifica se la bitmap è valida o meno ed è seguito da venticinque slot riservati alle pagine di bitmap.
I tre slot seguenti, cioe quelli dal 105 al 108 sui dischi attuali, contengono la data e l'ora in cui è avvenuta l'ultima scrittura sul disco. La data viene riportata con un solo valore, che esprime il numero di giorni tra il primo gennaio 1978 e la data presa in considerazione (ricordatevi di includere nel conto gli anni bisestili). L'ora è memorizzata con il numero di minuti trascorsi dalla mezzanotte e con il numero di secondi trascorsi nell'ultima ora.
I dodici slot seguenti sono destinati al nome del disco, che deve essere non più lungo di trenta caratteri. Le caselle dalla 121 alla 123 contengono la data e l'ora della creazione del disco (N.d.T.: o meglio della sua ultima formattazione); i tre successivi blocchi non vengono usati nel root block, mentre l'ultimo contiene un numero che indica il "sottotipo" del file (nel caso del root block 1).

User directory block

Uno user directory block è molto simile a un root block. I primi sei slot sono i medesimi, tranne per il fatto che lo slot uno contiene il numero del blocco stesso.
La hash table è del tutto uguale a quella nel root block. Al posto delle pagine di bitmap, ci sono dei bit di protezione e venticinque slot nei quali vengono memorizzati i vostri commenti al file. Gli slot dal 105 al 107 contengono l'ora e la data in cui il disco è stato formattato, nello stesso formato impiegato nel root block; gli slot dal 108 al 123 sono assegnati al nome della directory, mentre la casella 124 è come in precedenza dedicata alla catena hash. Lo slot 125 contiene, poi, il numero del blocco della directory superiore; il 126 non è usato e il 127 contiene "2" per indicare il sottotipo "user directory".

Se ponete un file in una directory, il numero del block header del file viene posto nella casella appropriata nel blocco della directory specificata invece che nel root block.
Supponiamo che abbiate creato una directory chiamata "Datafiles", che ha un valore hash di 32, e che abbiate posto "Filename" in tale user directory.
In tale situazione il root block conterrà il numero del blocco assegnato alla user directory nella casella 32 e la casella 59 di tale blocco conterrà il valore 884. Ecco perché dovete specificare l'intera successione di directory per accedere al file desiderato. In primo luogo viene letto il root block per trovare la user directory, quindi questa viene esaminata per individuare il numero del blocco di testa del file, da cui, poi, si ottengono i numeri dei blocchi in cui sono immagazzinati i dati. Se volete stampare le "key", cioè i numeri dei blocchi di testa dei vostri file, digitate il seguente comando:

LIST TO PRT: NomeDiDirectory KEYS

File header block

Un file header block contiene 512 byte, ovvero 128 slots, come quelli già considerati, ma in questo caso, ovviamente, essi hanno un differente impiego.
Le caselle zero e uno contengono l'informazione consueta (tipo del file e numero di questo blocco); lo slot 2 contiene il numero di blocchi usati dal file, il 3 precisa quanti, degli slot dal 6 al 77, sono stati usati per specificare i blocchi in cui il file è immagazzinato e il quarto slot contiene il numero del primo blocco di dati. Nello slot cinque, come al solito, è contenuto il checksum. I seguenti settantadue slot specificano i numeri dei blocchi usati per scrivere il file su disco.
I numeri dei blocchi compaiono in sequenza dal fondo della tavola, la casella 77, fino all'inizio, la casella 6. Nella maggior parte dei casi, i file sono immagazzinati su undici settori contigui; si saltano i successivi ventuno e poi si impiegano altri undici settori. La ragione di questa strategia particolare risiede nel fatto che così è possibile leggere una traccia intera alla volta, rendendo più efficiente il processo di lettura o di scrittura. Lo slot 80 contiene i bit di protezione mentre la lunghezza del file in bytes sta nello slot 81.
Ventidue slot sono dedicati ai vostri eventuali commenti; il resto del blocco è uguale a quello della user directory, eccezione fatta per la casella 126. Questa è infatti usata solo se la tavola dei blocchi contenenti i dati richiede più delle 72 caselle di questo blocco. Quando un file occupa più di settantadue blocchi, viene scelto un altro blocco, detto "extension block", per contenere gli altri numeri dei blocchi che compongono il file e il suo numero viene posto nello slot 126.

File list block

L'extension block (o file list block) inizia con i 6 slot che ormai conoscete: il tipo di file, il numero del blocco, il numero di slot usati per la tavola dei numeri dei blocchi del file, il numero del primo blocco di dati e il checksum. La tavola comincia anche qui in senso contrario, dallo slot 77 fino allo slot 6. Se si rivela necessario, può essere usato ancora un altro extension block. Le zone dedicate ai commenti e alla data non sono usate su questi blocchi e chiaramente non c'è catena hash; le ultime tre caselle, comunque, ospitano il numero del blocco della directory superiore, il numero del prossimo extension block (se c'è) e il sottotipo del file.

Data block

In questo tipo di blocchi, solo i primi sei slot sono occupati dalle informazioni per il DOS. In essi si trovano il tipo del blocco, il suo numero, il numero d'ordine del blocco (ad es. 5, se questo è il quinto blocco di dati del file), il numero di bytes di dati in questo blocco, il numero del prossimo blocco di dati di questo file e il checksum.
Tutto il resto del blocco è dedicato ai dati. Normalmente, tutti i blocchi di dati sono completamente pieni tranne l'ultimo. Al massimo in ogni blocco possono essere sistemati 488 bytes (128 - 6 = 122 slot da quattro byte ognuno). Nell'ultimo blocco potranno essercene di meno e il puntatore al prossimo blocco di dati (slot 4) conterrà zero.

Modificare un disco

A che cosa serve conoscere tutto questo?
Tanto per iniziare, può fornire delle risposte alle vostre domande sull'organizzazione del disco; ma se volete fare l'editing dei vostri dischi, allora è necessario che la catena hash e la hash table siano comprese a fondo.
Il recupero dei file è una parte di notevole importanza della manutenzione dei dischi e non si può recuperare alcun file se non avete chiara la struttura finora esposta.

Ultimamente, mi è arrivato un disco quasi completamente pieno, il cui root block conteneva un solo puntatore e questo puntatore era sbagliato: il disco era totalmente illeggibile. Ogni volta che si tentava di leggere una directory o di fare altre operazioni del tutto lecite, eravamo costretti a resettare il computer. Eppure siamo riusciti a recuperare l'intero disco!

Il disk editor che abbiamo usato è DiskEd, presente sul disco WACK incluso nel developers' package. Per recuperare il contenuto del disco, dovevamo in primo luogo localizzare gli header block di tutti i file che volevamo riportare alla luce.
La maggior parte delle user directory e dei blocchi header di tutti i file sono memorizzate nei blocchi tra l'880 e il 1100. I blocchi con numeri inferiori o superiori sono solitamente dei blocchi di dati. Con molta pazienza abbiamo esaminato i blocchi nell'intervallo prima precisato e abbiamo annotato i numeri di tutti gli header block trovati e i valori hash dei corrispondenti nomi di file; abbiamo quindi letto il blocco 880 nel quale abbiamo ricostruito, sulla base delle informazioni prima ottenute, la struttura della directory principale. Per far questo abbiamo inserito ogni numero di header block nello slot associato al valore hash corrispondente al nome del file, secondo la struttura esposta all'inizio di questo articolo.
Una volta terminata questa operazione, abbiamo ricalcolato il checksum del root block e finalmente abbiamo riscritto il root block stesso sul disco. L'operazione di salvataggio del disco era così conclusa; la richiesta della directory veniva ora espletata senza alcun problema e il contenuto del disco era a tutti gli effetti normale.

DiskEd è stato uno dei primi programmi di editing dei dischi. Ora è senz'altro sorpassato; l'operazione di recupero di file da un disco rovinato può essere effettuata in maniera del tutto automatica da programmi come DiskDoctor o come DiskSalv (Fish 20). Ci sono poi diversi disk-editor di pubblico dominio o shareware, come ad esempio Sectorama (Fish 108) e DiskX (Fish 71), che possono reggere senza problemi il confronto con programmi commerciali simili, e un gran numero di utility sempre di pubblico dominio che possono rivelarsi molto utili nelle esplorazioni della struttura dei dischi.

(segue da pagina 69)

Il Manx C 3.6 e l'SDB

Una differenza importante tra DB e SDB è che quest'ultimo non sta in "background", indipendente dal programma che state correggendo. Come risultato, non potete fare il debugging di sotto-task nella maniera di DB, benché sia disponibile un modo speciale per operare su library o device scritti dall'utente.

Documentazione

Il manuale di SDB è costituito da un piccolo raccoglitore con tre anelli e rappresenta poco di più di una dettagliata guida di riferimento. Quasi nulla è presente per quanto concerne esempi o introduzioni guidate, ma la carenza è parzialmente compensata da un demo piuttosto completo sul disco. Alcune descrizioni dei comandi sono confuse, inaccurate o tutt'e due (qualche volta è difficile stabilirlo). Il comando "p" per la stampa formattata è il più barocco tra quelli di SDB, richiedendo nove pagine di una stringata descrizione delle sue molte opzioni. Per quanto posso dire, dopo lunghi studi del manuale e sperimentazioni sul programma, la maggior parte delle opzioni non funziona. O forse non ho proprio capito.

Ciononostante...

Chiunque programmi intensivamente con il compilatore C Manx si accorgerà molto probabilmente che l'impiego di SDB giustifica più che ampiamente il suo acquisto.

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

Cognome .....

Nome .....

Via .....

Cap. ..... Città .....

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

Firma .....

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

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

IL SIGNIFICATO DEI COLORI
I Messaggi di Start-Up

di Betty Clay

Non è raro che un computer cerchi di comunicare con l'utente durante le procedure di start-up. Ricordate le lucine che lampeggiavano sui drive 4040 e 8050? Sul mio drive 8050, due lampeggi del LED verde indicavano che tutto procedeva regolarmente; più di due indicavano un problema e in tal caso il numero di lampeggi poteva essere di grande aiuto a un eventuale riparatore.
Un'altra maniera mediante la quale le prime macchine Commodore comunicavano con noi era segnalando sullo schermo, durante la fase di start-up, il numero di byte liberi. Se tale valore era sbagliato, potevamo immediatamente accorgerci che una zona di RAM si era rifiutata di accogliere dei dati e dal numero apparso potevamo risalire al chip di RAM guasto.
L'Amiga ha un sistema diagnostico piuttosto elaborato, che risulta utile, però, solo se sappiamo come interpretarne i messaggi.

La procedura di start-up

Quando un Amiga viene acceso, inizia a eseguire un insieme piuttosto lungo e articolato di routine di inizializzazione, cercando diligentemente di informarvi sull'esito dei singoli passi di tale procedura. C'è, tuttavia, un piccolo problema: la Commodore si è "dimenticata" di spiegare come interpretare gli eventuali segnali di anomalia.

Prima di entrare nel dettaglio, vi presentiamo la lista delle operazioni compiute durante la fase di start-up:

1. Viene interamente pulita la RAM.
2. Vengono disabilitati il DMA e gli interrupt durante i test iniziali.
3. Si pulisce lo schermo.
4. Si controlla che il 68000 stia effettivamente funzionando.
5. Il colore dello schermo viene cambiato se il test è andato bene.
6. Viene operato un checksum di tutte le ROM.
7. Si cambia il colore dello schermo in caso di esito positivo del test.
8. Si dà inizio allo start-up del sistema.
9. Viene controllata la presenza della RAM in $C00000 e, se possibile, vi si fà puntare SYSBASE.
10. Si controlla tutta la Chip RAM.
11. Il colore dello schermo cambia per segnalare che la Chip RAM è OK.
12. Si controlla che il software stia procedendo senza problemi.
13. Si cambia il colore dello schermo in caso di esito positivo del test.
14. Si appronta la Chip RAM per farle ricevere i dati.
15. Vengono aggiunte al sistema le librerie.
16. Si controlla la presenza di RAM aggiuntiva e, eventualmente, la si aggiunge a quella già disponibile.
17. Vengono attivati il DMA e gli interrupt.
18. Parte un task di default.
19. Si accerta la presenza del 68010, del 68020 e/o del 68881.
20. Si controlla se c'è una Exception e, in caso affermativo, si "resetta" il sistema.

Messaggi in Technicolor!

Mentre ha luogo questa procedura, Amiga vi invia messaggi tramite i colori dello schermo.
Se tutto procede regolarmente, si avrà questa successione di colori:

Grigio scuro	Un primo test sullo hardware ha avuto esito positivo. Il 68000 funziona e i registri possono essere letti.
Grigio chiaro	Il software procede senza problemi.

Bianco	Tutti i test di inizializzazione sono stati superati.

Se, però, qualcosa non va per il verso giusto, potreste vedere:

Rosso	Se è stato rilevato un errore nelle ROM.
Verde	Se c'è un'anomalia nella Chip RAM.
Blu	Se si è evidenziato un errore nei chip custom.
Giallo	Se il 68000 ha trovato un errore, prima che il sistema consueto di rilevazione degli errori (il Guru, insomma) fosse stato approntato.

Il più frequente di questi errori sembra essere l'errore in Chip RAM.
Solamente questa settimana, ho visto per la prima volta un Amiga 500 far mostra di un bello schermo verde brillante a causa di una espansione RAM inserita male.
Il problema è sparito una volta che la board è stata inserita nella maniera corretta. Finora non ho mai visto uno schermo rosso, blu o giallo.

I messaggi della tastiera

Ho scoperto che la tastiera di Amiga non è un oggetto così ordinario come pensavo.
Infatti contiene un processore dedicato (il 6500/1, prodotto dalle ditte Rockwell, NCR e MOS Technology) dotato di 2K di ROM, 64 bytes di RAM, quattro porte di I/O da otto bit ognuna e un oscillatore quarzato da 3 MHz.
Inoltre, tutte, eccetto le primissime macchine, hanno un "timer di guardia", che resetta il processore della tastiera se si accorge che la tastiera non è stata esaminata per un intervallo di tempo superiore ai 50 millisecondi.

Nel caso, perfettamente ammissibile, in cui la tastiera venga collegata al calcolatore già acceso, questa deve, al momento della connessione, eseguire un test su se stessa per verificare il suo funzionamento.

In generale, comunque, la tastiera rimane sempre attaccata all'unità centrale e questo test ha luogo mentre noi siamo guardando lo schermo, inserendo dischi, ecc.

Il test automatico della tastiera è suddiviso in quattro fasi. Viene per prima cosa calcolato e controllato il checksum delle ROM; quindi, vengono controllati i 64 byte di RAM e il timer. A questo punto la tastiera deve sincronizzarsi opportunamente con il calcolatore.
Ciò viene ottenuto inviando degli "uno" piuttosto distanziati nel tempo l'uno dall'altro fino a quando viene ricevuto un impulso di risposta dal computer.
Quando questo avviene, la tastiera informa il computer dell'esito dei test attraverso un codice particolare.

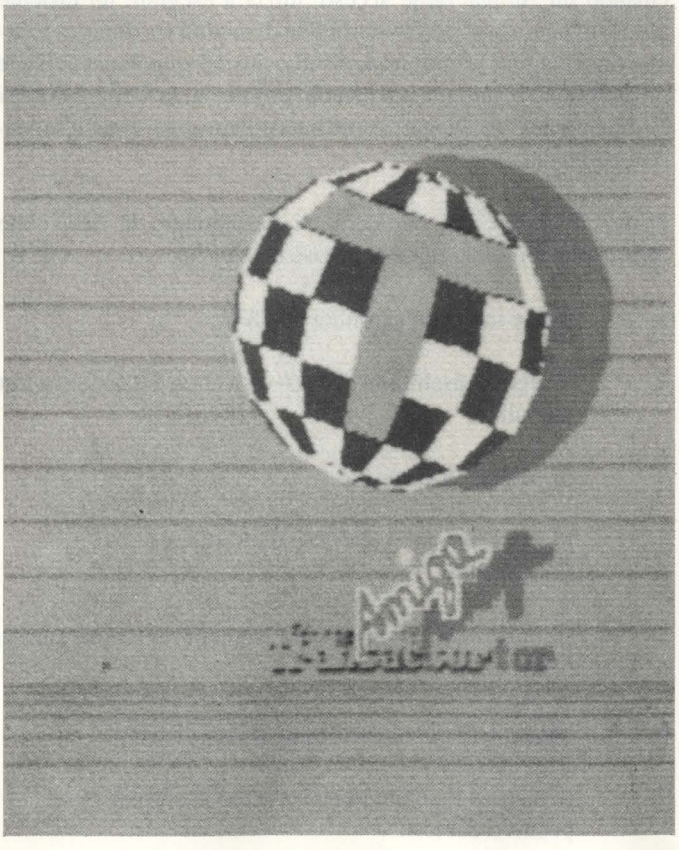
In caso di guasto

Dopo aver informato il computer del cattivo esito dei test, la tastiera tenta di comunicare all'utente la presenza di anomalie tramite il LED del CAPS LOCK secondo la seguente modalità:

Un lampeggio	La ROM della tastiera è guasta.
Due lampeggi	La RAM della tastiera è guasta.
Tre lampeggi	Il timer di controllo non funziona.
Quattro lampeggi	C'è un corto circuito tra due file di tasti o in uno dei sette tasti speciali di controllo.

L'ultimo controllo, per la verità, non era stato ancora implementato al momento in cui il mio ROM Kernel Manual venne stampato, ma era solo un progetto futuro.
Sarebbe certo strano che l'utente digitasse qualcosa durante i test iniziali, ma nel caso in cui venisse premuto qualche tasto, i codici corrispondenti verrebbero inviati al computer, seguiti dal codice di "terminate key stream" (termine del flusso di tasti); infine, il LED del CAPS LOCK si spegnerebbe, indicando la fine della fase di start-up della tastiera.

Se dovessimo essere così sfortunati da avere qualche problema, questi messaggi potranno aiutare noi o il nostro riparatore di fiducia a restituire la buona salute al nostro Amiga.



Warrior cycles: i programmi combattono in un'arena su schermo

di Rico Mariani

...l'Amiga è ovviamente il perfetto candidato per giochi come questo...

Rico Mariani è stato un programmatore part-time alla Comspec Communications Inc. dai tempi del PET 8K e un hacker Amiga fin dal primo momento. E' uno studente di Scienze dell'Informazione dell'Università di Waterloo e passa la maggior parte del suo tempo al Centro Scientifico dell'Ontario scrivendo software per sistemi interattivi sull'Amiga e su altri computer basati sul 68000.

Diversi anni fa lessi di un programma disponibile per Apple II chiamato "Robot Wars". Il gioco consisteva nel far programmare a ogni giocatore un robot che poi doveva essere lasciato libero per combattere contro tutti gli altri. Questo torneo sembrerebbe il miglior test per verificare le propria abilità di programmatore: il proprio programma combatte uno scontro mortale contro un altro. Un gioco di ispirazione simile, apparso ultimamente su "Le Scienze", è "Core Wars". In Core Wars ogni giocatore crea un piccolo programma con speciali istruzioni di tipo assembler allo scopo di cancellare dalla memoria di una macchina virtuale, dove vengono caricati i contendenti, i propri avversari. La bellezza di Core Wars risiede nel fatto che i programmi sono così semplici e facili da scrivere che le possibili strategie sono praticamente infinite.

Una macchina multitasking come l'Amiga è ovviamente il candidato perfetto per giochi come questo, specialmente per le ottime possibilità grafiche usate per visualizzare il gioco in tutte le sue fasi. Quindi, ecco il mio gioco per competizioni tra programmi multipli ideato per l'Amiga: Warrior Cycles.

Warrior Cycles è un gioco basato su "Light Cycles" del film "Tron". Ogni giocatore scrive un programma che controlla gli spostamenti di un Cycle in un'arena le cui dimensioni non sono note finché il gioco non comincia. Le regole della sfida sono semplici: mentre un Cycle si muove lascia una scia dietro di sé. Un Cycle viene eliminato quando coccia contro un altro Cycle oppure contro a una delle scie lasciate da qualche Cycle (al limite anche da sé stesso). Il gioco continua finché non rimane un unico Cycle. Eventualmente, se gli ultimi giocatori vengono eliminati tutti nello stesso momento, la partita viene ritenuta nulla.

Warrior Cycles è costituito da un programma principale (server) e da uno o più programmi che si sfidano (giocatori). Il server crea l'arena, usando uno schermo Intuition, nella quale si sfideranno i programmi. I giocatori sono programmi separati scritti dagli sfidanti (umani) nel proprio linguaggio preferito. Il giocatore controlla il Cycle nell'arena chiedendo informazioni al server su che cosa ci sia al momento sullo schermo e fornendo in risposta la direzione in cui si vuole muovere nel proprio turno. Il server viene eseguito da CLI per primo (usando il comando RUN), seguito da ognuno dei giocatori. Il server attende che un certo numero di giocatori prenda parte alla competizione prima di iniziare.

Le linee, ognuna di un colore diverso, lasciate nell'arena indicano i percorsi seguiti dai diversi Cycle. Il numero di partite vinte da ogni Cycle (appena una termina ne viene iniziata subito un'altra) è visualizzato nella barra del titolo con lo stesso colore di ogni Cycle. Le partite hanno un'andatura molto spedita e sono divertenti da vedere. Si può verificare velocemente quale sia il giocatore migliore dopo pochi turni dando uno sguardo ai punteggi. Nel frattempo, nello schermo del Workbench, ogni giocatore ha creato una propria finestra sufficientemente piccola e spaziata rispetto alle altre in modo che possano essere viste tutte contemporaneamente. I titoli delle finestre, all'interno delle quali viene visualizzato il colore del giocatore (giallo, blu ecc.), mostrano i nomi dei giocatori. Nelle finestre si trova anche un resoconto costantemente aggiornato delle partite vinte, perse e di quelle finite pari. Si può ritirare un giocatore dalla gara attivando il gadget per la chiusura della finestra, mentre se ne può aggiungere uno lanciandolo direttamente da CLI. In effetti i giocatori non possono lasciare o entrare nel bel mezzo di una partita, ma devono attendere finché non termina. Anche il server dispone di un gadget per la chiusura della finestra (close gadget) che causa la fine prematura della sfida. Quando il server viene fatto terminare manda un messaggio a tutti i giocatori indicando la fine della sfida e attende poi che rispondano tutti.

Mentre il torneo è in svolgimento, si può vedere il gioco vero e proprio sullo schermo dell'arena oppure si può ripristinare lo schermo Workbench, usando i gadget appropriati, per controllare chi sta partecipando e come stanno andando le cose. Eventualmente si può partecipare alla sfida giocando in modo interattivo: infatti è stato fornito un giocatore che non possiede una strategia propria ma che segue le indicazioni fonitegli da un joystick. In questo modo, altre a giocare contro i giocatori "robot", si può testare il funzionamento di una propria creatura.

La versione attuale di Warrior Cycles supporta un massimo di sei giocatori. Non si tratta di una limitazione intrinseca del programma, ma esiste perché:

(1) viene usato uno schermo con tre bit plane (8 colori),

(2) più giocatori (programmi) sono in esecuzione nello stesso

momento, meno tempo CPU può essere a loro dedicato, quindi più lungo deve essere ogni turno.

Una maggior lentezza porta ad avere un torneo noioso, quindi il numero dei giocatori deve essere tenuto basso a beneficio degli spettatori umani. Così com'è si possono avere partite veloci e si può lasciare sufficiente tempo ai giocatori per pensare tra una mossa e l'altra.

Quando viene mandato in esecuzione il server, possono essere fornite tre opzioni: il numero di giocatori necessari perché una partita sia cominciata, la quantità di tempo tra una mossa e l'altra, e il numero di round da disputare. E' importante usare il numero corretto di giocatori che devono attivi prima che una partita cominci, per evitare di iniziare prima che si siano "registrati" tutti. Infatti, se la sfida comincia senza che tutti siano presenti, qualche giocatore deve aspettare il prossimo round prima di entrare in competizione. La quantità di tempo tra una mossa e l'altra è specificata in millisecondi per giocatore; ciò evita di dover modificare questo valore ogni volta che cambia il numero dei partecipanti. Questo valore è solitamente posto a 20 se si vuole una gara spedita; un valore più alto significa poter utilizzare programmi con strategie più complesse poiché ognuno dispone di un tempo maggiore per pensare. Torneremo più tardi sui valori da assegnare in fase di attivazione del server.

Di seguito è riportata la tipica linea di comando per invocare il server, richiedendo una sfida in dieci partite, con tre giocatori e a venti millisecondi per turno (la categoria "pesi leggeri"):

```
run Cycle p3 g10 t20
```

I valori di default per p, g e t sono rispettivamente 2, 0 e 6. Questi valori sono adatti per sperimentare il gioco con due giocatori banali; normalmente si preferisce definire da sé i valori appropriati. Il richiamo dei giocatori è molto semplice:

```
run player1
run player2
run player3
```

Quando si esegue "run Cycle", il server attiva immediatamente il proprio schermo, rendono invisibile quello del Workbench. Occorre quindi modificare l'ordine degli schermi per impartire i successivi comandi per l'attivazione dei giocatori. Il miglior metodo, alla fine, è mettere i comandi precedenti in un file, chiamato per esempio "torneo.cmd", in modo che si possa dare inizio alla sfida con un solo comando del tipo:

```
execute torneo.cmd
```

Dettagli tecnici

Il server imposta lo schermo grafico e tiene traccia della posizione e della direzione di ogni giocatore. Ogni giocatore (un "cliente" per il server) che desidera partecipare al gioco deve mandare un messaggio a una porta pubblica di proprietà del server. Una volta ricevuto il messaggio, il server accetta il nuovo giocatore e gli assegna un colore. Il cliente successivamente manda ulteriori messaggi per indicare la server quale direzione intende prendere; il server risponde informando il cliente che la mossa è stata completata oppure che è stato eliminato o dichiarato vincitore. Tutto il meccanismo per la suddivisione del tempo è effettuato dall'Exec. Quando si manda in esecuzione il server e i giocatori per mezzo del comando RUN, uno riconosce la presenza degli altri e dà inizio alle danze.

Un beneficio di questo metodo di implementazione risiede nel fatto che ogni giocatore può essere scritto con il linguaggio che l'autore preferisce. Fin tanto che i giocatori mandano al server i messaggi nel formato richiesto, vengono aggiornati e informati sul proprio stato indipendentemente dal linguaggio che è stato usato.
In aggiunta al programma server, viene fornito un insieme di appropriate routine di interfacciamento che permettono di non conoscere tutti i dettagli del passaggio di messaggi che intercorre con il server. Le routine sono state scritte in C ma possono essere facilmente convertite per il Modula-2, l'Assembler o qualunque altro linguaggio si desideri.

Routine per i programmi giocatori

Ciò che segue è una breve descrizione di ogni funzione. Se si desidera scrivere un programma giocatore, bisogna porre molta attenzione nell'esame dell'implementazione di ogni funzione. A fianco di ogni funzione è posta una nota indicante se la funzione causa la spedizione di qualche messaggio: si tratta di un dettaglio importante poiché il mandare un messaggio non è operazione che viene portata a termine immediatamente.

```
stato = Register(colore)  (manda un messaggio
                                al server)
int  *colore;
```

Questa funzione cerca di comunicare al server che il giocatore vuole partecipare alla sfida. L'intero referenziato dall'argomento colore sarà riempito con il codice del colore di quel giocatore. Se la registrazione non ha avuto successo viene restituito un valore diverso da zero. Occorre usare la funzione Register prima di utilizzare una delle altre.

```
Retire()      (manda un messaggio al server)
```

Questa funzione informa il server che si ha intenzione di abbandonare il gioco; può essere richiamata solo dopo che ci si è registrati con successo. Se si vuole partecipare ancora alla gara occorre registrarsi (con la funzione Register()) un'altra volta.

```
risultato = Inquire(x,y)  (non manda nessun
                            messaggio al server)
int  x,y;
```

Questa funzione permette di chiedere se la cella di coordinate (x,y) sia o meno occupata: restituisce TRUE se lo è, FALSE se è libera. Tutte le celle occupate dalla scia di un Cycle e tutte quelle poste all'esterno dell'arena sono considerate occupate. In altre

parole, se un giocatore si muove in una cella per cui Inquire() ha restituito il valore TRUE, significa che è andato a sbattere contro la scia di un Cycle oppure contro al muro dell'arena ed è di conseguenza stato eliminato.

```
risultato = Look(dir)  (non manda nessun
                        messaggio al server)
int dir;
```

Questa funzione restituisce il valore TRUE se il quadrato posto nella direzione indicata è ostruito. Il parametro dir può assumere uno dei valori DIR_UP, DIR_RIGHT, DIR_DOWN o DIR_LEFT come sono definiti nel file "Cycles.h". Riferirsi anche a GetInfo().

```
immagine = GetScreenMem()  (non manda nessun
                            messaggio al server)
```

Questa funzione restituisce un pointer alla memoria che descrive la bitmap dell'arena attuale. Non si deve assolutamente scrivere in quest'area di memoria visto che viene condivisa da tutti i Cycle in gara. Una volta che si possiede un puntatore all'immagine dell'arena si può scoprire quanto sia grande e quali quadrati siano già occupati. Questa possibilità può essere usata, ad esempio, trovare dove si trovino i propri avversari facendo una copia della bitmap per confrontarla con la nuova bitmap che si è formata dopo che tutti i Cycle si sono mossi. Riferirsi a "Cycles.h" per conoscere il formato esatto di una struttura IMAGE.

```
GetInfo(x,y,dir)  (non manda nessun messaggio
                                    al server)
int *x, *y, *dir;
```

Questa funzione pone negli interi puntati da x, y e dir i valori relativi alla propria posizione e alla direzione nella quale ci si sta muovendo. La direzione può avere uno dei valori DIR_UP (su), DIR_DOWN (giù), DIR_RIGHT (destra) e DIR_LEFT (sinistra). In Cycles.h sono definite due macro, chiamate TURN_RIGHT (gira a destra) e TURN_LEFT (gira a sinistra), per trasformare le direzioni. TURN_RIGHT(dir) restituisce la direzione che si prenderebbe girando a destra rispetto alla direzione specificata. TURN_LEFT agisce nello stesso modo supponendo però una svolta a sinistra.

```
stato = Direction(dir)  (manda un messaggio al
                                    server)
int dir;
```

Questa funzione informa il server che si desidera che la prossima mossa sia nella direzione indicata dall'argomento dir. Questa funzione restituirà il controllo solo dopo che la mossa è stata eseguita. Occorre notare che questo è un sistema di gioco real-time e quindi il server si aspetta di essere informato della nuova direzione a intervalli regolari. Se si passa troppo tempo a pensare in quale direzione muoversi e non si riesce a comunicare la nuova direzione, ci si continuerà a muovere nella direzione precedente; se ciò vuol dire sbattere contro un muro, peggio per voi. Se si rende il proprio programma troppo veloce, gli avversari avranno più tempo per pensare mentre voi state aspettando che il server muova in vostra vece. Quindi occorre stare attenti: non troppo lento ma neanche troppo veloce.
Il valore restituito dalla funzione può essere uno dei seguenti:

```
STATUS_NEED_NEW     tutto OK, pronto
                    per la prossima
                    mossa
STATUS_LOSER        tutto male, siete
                    stati eliminati
STATUS_WINNER       siete stati
                    dichiarati
                    vincitori
STATUS_REMOVED      il server sta
                    terminando,
                    gioco sospeso
```

Nota: non bisogna cercare di mandare un messaggio al server dopo che si è ricevuto STATUS_REMOVED.

```
stato = Await()  (manda un messaggio al
                                    server)
```

Questa funzione comunica al server che ci si desidera iscrivere alla prossima partita. Il server successivamente comunicherà quando si deve ricominciare. Occorre chiamare questa funzione dopo che si è stati dichiarati WINNER (vincitori) o LOSER (sconfitti). Se non si desidera continuare occorre invece chiamare Retire(). Il valore restituito dalla funzione può essere uno di quelli restituiti da Direction(), eccetto quelli che indicano che si è vinto o che si è perso: infatti, oltre ad attendere la vostra prossima mossa, il server potrebbe anche aver terminato l'esecuzione.

Supporto alla programmazione

Per semplificare la scrittura di programmi-giocatori è stato creato un semplice programma che gestisce e si prende cura dell'apertura, della registrazione e di tutte queste amenità. Tutto ciò che si deve fare è creare tre funzioni e dare un nome al proprio giocatore, per disporre di un programma funzionante. Basta poi porre queste funzioni in un modulo, effettuare il link con il programma gestore, per ottenere il proprio giocatore pronto per combattere nell'arena Warrior Cycle.

```
int strategy_init();
```

Con questa funzione si devono effettuare tutte le inizializzazioni richieste dal proprio giocatore. Se restituisce un valore diverso da zero, il programma gestore termina l'esecuzione, rimuovendo il programma-giocatore dal sistema.

```
int strategy();
```

E' il nucleo centrale del programma. Qui si devono richiamare le routine per interrogare il server su quale sia la situazione per poi decidere quale direzione intraprendere. Deve restituire uno dei valori DIR_UP, DIR_DOWN, DIR_RIGHT o DIR_LEFT. Se restituisce un valore negativo termina l'esecuzione.

```
int strategy_finish();
```

E' la funzione con cui si devono effettuare tutte le operazioni di pulitura, tipo la restituzione di memoria allocata durante la fase di inizializzazione.

Occorre anche definire la seguente variabile:

```
char player_name[];
```

E' un array di caratteri che deve contenere il nome del proprio programma-giocatore.

Il modo migliore di verificare il funzionamento del tutto, è di dare uno sguardo ad alcuni dei programmi giocatori già ideati.

Player	*Nome*	*Personalità*
1	Mr.Right	Se davanti c'è un problema, gira a destra
2	Dizzy	Cerca sempre di girare a destra, se può
3	Vertigo	Preferisce andare su/giù piuttosto che destra/sinistra
4	Round the clock	Guarda in ogni direzione e prende quella più libera
5	Keep'em flying	Come il precedente ma preferisce mantenere la direzione attuale
6	Not so Dizzy	Guarda avanti tre mosse e sceglie la direzione migliore
7	Look B4 You Leap	Guarda avanti 4 mosse, gli piace andare dritto
8	Joystick	Segue le indicazioni del joystick

I primi tre sono chiamati pesi piuma: decidono la propria mossa dando uno sguardo ai quadrati a loro immediatamente adiacenti. Rispondono al server molto velocemente. I due successivi sono pesi leggeri: si guardano in giro un po' in una certa direzione. I giocatori 6 e 7 sono pesi medi: ispezionano tutte le loro vicinanze prima di decidere. L'ultima classe (i pesi massimi), per la quale non sono stati preparati esempi, dovrebbe comprendere dei giocatori che considerano anche la posizione dei concorrenti. La distinzione in classi è importante perché la frequenza con cui il server effettua le mosse non è fissa. Se la frequenza è troppo alta, i pesi medi non hanno abbastanza tempo a disposizione per pensare e vengono sopraffatti dai pesi leggeri, che impiegano meno tempo per generare una mossa. Eventualmente, una frequenza ancora più alta porterebbe i pesi piuma a dominare la scena.

Generalmente si dovrebbero usare i seguenti valori a seconda del tipo di classe:

piuma	10 ms/giocatore
leggeri	20 ms/giocatore
medi	50 ms/giocatore
massimi	200 ms/giocatore

La scrittura di un buon peso massimo appare come una vera sfida.

Si dia inizio alla sfida

Eccoci giunti alla fine della trattazione: è ora di mettere in campo i propri campioni per una serie di sfide all'ultimo respiro. Ognuno può organizzare eventuali tornei come meglio crede, ma sarebbe opportuno mantenere tutti lo stesso approccio per poter confrontare i risultati in un secondo momento. Ferma restando la divisione nelle diverse classi proposte bisognerebbe evitare di permettere l'uso di tecniche volte a modificare le proprie possibilità rispetto a quelle degli altri contendenti. Per esempio non dovrebbe essere possibile modificare la priorità del proprio task o di quelli avversari, non si dovrebbe poter creare altri task che eseguano una serie di operazioni per il giocatore (aumentando di fatto il tempo di CPU a disposizione) e non si dovrebbe poter modificare le strutture dati condivise dal server. Le sfide tra più di due avversari potrebbero essere giocate essenzialmente in due modi: tutti contro tutti in un'unica sfida con un certo numero di round, oppure con un girone all'italiana. Il primo previligerebbe lo spettacolo, ma il risultato sarebbe fortemente influenzato anche dalla fortuna, mentre il secondo fornirebbe risultati più attendibili, fornendo però uno spettacolo visivo più limitato.

A voi la scelta, e che vinca il migliore!

Listato 1: I file sorgente del server di Warrior Cycles. Tutti i tre moduli devono essere compilati e linkati per produrre il programma server. Se state usando il Manx bisogna compilare con l'opzione interi a 32 bit. Gli utenti del Lattice C non dovrebbero avere problemi anche se non è stato testato con questo compilatore.

Listato 1a

"cycles.c" - La parte centrale del server, dove arrivano i messaggi dei giocatori e vengono trattati.

```
/*
 * Programma principale di Cycles... Attende che
 * arrivino i messaggi e agisce di conseguenza
 */
#include <intuition/intuition.h>
#include <exec/memory.h>
#include <devices/timer.h>
#include <libraries/dos.h>
#include "cycles.h"

extern int Enable_Abort;
extern BYTE scr_image[];

int xdir[] = {  0,  1,  0, -1 };   /* su, destra, giù, sinistra */
int ydir[] = { -1,  0,  1,  0 };   /* su, destra, giù, sinistra */
int TIME   = 6;
int PLAYERS = 2;
int GAMES   = 0;
int scores[MAX_COLOURS];

IMAGE  scr_data;      /* dati dell'immagine sullo
                                          schermo*/
```

```
msg->status = STATUS_REMOVED;
        if (msg->cycle) num_players—;
          ReplyMsg(msg);
      }
    }
    g_finish();

    /* Qui si pulisce la porta prima di chiuderla.
                                  E' importante chiamare
     * prima Forbid() in modo che non arrivi nessun
                                  messaggio tra l'ultimo
     * momento in cui ci accediamo e quando la
                                  cancelliamo
     */

    Forbid();
    while (msg = (ORDER *)GetMsg(sPort)) {
      msg->status = STATUS_REMOVED;
      ReplyMsg(msg);
    }
    DeletePort(sPort);
    Permit();
    exit(0);
}
```

Listato 1b

"dobikes.c"

```
/*
 * Muove i cycle sullo schermo e modifica il loro stato
 */

#include "cycles.h"

extern int xdir[], ydir[];
extern int num_players;
extern int num_waiting;
extern int num_alive;

int ties;

move_cycles(list)
LIST *list;
{
        register CYCLE *c, *c2;
        int na;

        na = num_alive;

        c = (CYCLE *)list->lh_Head;
        if (c->node.ln_Succ) {          /* gira la lista
                                           in modo che */
        Remove(c);                      /* ogni cycle abbia
                                           un turno in cui */
        AddTail(list,c);        /* gli viene risposto per
                                           primo */
        }
        c = (CYCLE *)list->lh_Head;

        while (c2 = (CYCLE *)c->node.ln_Succ)
        {
                switch (c->status) {

                case STATUS_GOT_MOVE:
                case STATUS_NEED_NEW:
                        link_pix(c->colour,c->x,c->y,
                                c->dir);
                        c->x += xdir[c->dir];
                        c->y += ydir[c->dir];
                        if (rpix(c->x,c->y)) {
                                c->status = STATUS_LOSER;
        num_alive—;
        if (num_alive == 0) {
        ties++;
                                draw_scores();
                                flash_colour(0);
        }
}
else {
        if (na>1)
                                c->status
                                STATUS_NEED_NEW;
else {
                                scores[c->colour]++;
                                draw_scores();
                                c->status =STATUS_WINNER;
                                flash_colour(c->colour+3);

        num_alive—;
        }
        wpix(c->colour,c->x,c->y,c->dir);
        }

        if (c->msg) {
                                c->msg->x     = c->x;
                                c->msg->y     = c->y;
                                c->msg->dir   = c->dir;
                                c->msg->status = c->status;
                                ReplyMsg(c->msg);
                                c->msg = NULL;
        }
        break;

        default:
        break;
        }
        c = c2;
        }
}


free_cycles(list)
LIST *list;
{
        register CYCLE *c,*c2;

        c = (CYCLE *)list->lh_Head;

        /* Controlla tutti i cycles... */

        while (c2 = (CYCLE *)c->node.ln_Succ) {

                /* Se non c'e' nessun messagio in
                                                attesa */

                if (c->msg) {
                        /* informa il cliente che è
                                        stato rimosso */
                        c->msg->status =
                                STATUS_REMOVED;
                        ReplyMsg(c->msg);
                        c->msg = NULL;
                        num_players—;
                }
                        /* Pulisce il cliente... */
```

```
                Remove(c);
                FreeMem(c,sizeof(CYCLE));
                c = c2;
        }
}

restart_cycles(list)
LIST *list;
{
        register CYCLE *c,*c2;

        c = (CYCLE *)list->lh_Head;

        while (c2 = (CYCLE *)c->node.ln_Succ) {
                if (c->status == STATUS_AWAITING) {
                        c->x      = rand()%(X_SIZE-
                                10)+5;
                        c->y      = rand()%(Y_SIZE-
                                10)+5;
                        c->dir    = DIR_RIGHT;
                        c->status      =
                                STATUS_NEED_NEW;
                        c->msg->x      = c->x;
                        c->msg->y      = c->y;
                        c->msg->status = c->status;
                        c->msg->dir    = c->dir;
                        wpix(c->colour,c->x,c->y,c->dir);

                        ReplyMsg(c->msg);
                        c->msg = NULL;
                }
                c = c2;
        }
        num_waiting = 0;
        num_alive = num_players;
}
```

Listato 1c

"graphic.c"

```
/*
 * Qui c'è tutto il codice per la grafica...
 *
 */

#include <exec/types.h>
#include <intuition/intuition.h>
#include "cycles.h"

#define INTUITION_REV 29
#define GRAPHICS_REV 29
#define X_BYTES ((X_SIZE/8)+!!(X_SIZE%8))

struct NewScreen newscreen = {
        0,0,                    /* Angolo in alto a
                                        sinistra */
        320,200,                /* larghezza, altezza */
        3,                      /* profondità */
        0,1,                    /* DetailPen, Block
                                        Pen */
        0,                      /* viewmode */
        CUSTOMSCREEN,                   /* tipo */
        NULL,                   /* font */
        (UBYTE *)" Cycles",     /* titolo */
        NULL,                   /* Gadget */
        NULL                    /* CustomBitMap*/
};

struct NewWindow newwindow = {
        0,0,                    /* Angolo in alto a
                                        sinistra */
        15,10,                  /* larghezza, altezza */
        0,1,                    /* DetailPen,
                                        BlockPen */
        CLOSEWINDOW,            /* Flag IDCMP */
        WINDOWCLOSE,            /* Flag */
        0,                      /* Primo Gadget */
        0,                      /* checkmark */
        0,                      /* titolo */
        0,                      /* Lo schermo verrà
                                   riempito più tardi */
        0,                      /* bitmap */
        0,0,                    /* Larghezza e
                                        altezza minima */
        -1,-1,                  /* Larghezza e altezza
                                        massima */
        CUSTOMSCREEN                    /* tipo */
};

UWORD scr_colours[ ] = {
        0x000, 0xf80, 0x002, 0xf00,
        0x00f, 0xfff, 0x0f0, 0xff0
};

BYTE scr_image[ X_BYTES * Y_SIZE];

struct IntuitionBase    *IntuitionBase;
struct GfxBase                  *GfxBase;
struct Screen                   *war_screen;
struct Window                   *war_window;
struct RastPort         *rp;
struct Screen                   *OpenScreen();
struct Window                   *OpenWindow();
void                            *OpenLibrary();

g_init()
{
        IntuitionBase =
OpenLibrary("intuition.library",INTUITION_REV);
        if (!IntuitionBase) {
                g_finish();
                printf("Non riesco ad aprire Intuition\n");
                return(1);
        }

        GfxBase =
OpenLibrary("graphics.library",GRAPHICS_REV);
        if (!GfxBase) {
                g_finish();
                printf("Non riesco ad aprire la libreria
                                                grafica\n");
                return(1);
        }
        war_screen = OpenScreen(&newscreen);
        if (!war_screen) {
                g_finish();
                printf("Non riesco ad aprire lo
                                        schermo\n");
                return(1);
        }
        LoadRGB4(&war_screen->ViewPort,
scr_colours, 8);

        rp = &war_screen->RastPort;
```

```
        newwindow.Screen = war_screen;

        war_window = OpenWindow(&newwindow);
        if (!war_window) {
                g_finish();
                printf("Non riesco ad aprire la
                                        finestra\n");
                return(1);
        }
        return(0);
}

g_restart()
{
        int i,j;
        for (i=0;i<X_BYTES*Y_SIZE;i++) scr_image[i] =
0;

        SetAPen(rp,0);
        RectFill(rp,0,10,319,199);
        SetAPen(rp,1);
}

g_finish()
{
        if (war_window) CloseWindow(war_window);
        if (war_screen) CloseScreen(war_screen);
        if (GfxBase) CloseLibrary(GfxBase);
        if (IntuitionBase) CloseLibrary(IntuitionBase);
}

static int power[] = {1,2,4,8,16,32,64,128};

        rpix(x,y)
{
        if (x<0 || y<0 || x>=X_SIZE || y>=Y_SIZE)
                return(1);

        return(scr_image[(x>>3) + y*X_BYTES] &
        power[x&7]);
}

typedef struct {
        int x,y;
} PAIR;

PAIR old_data[] = { {4,7},{0,4},{4,0},{7,4} };
PAIR new_data[] = { {4,0},{7,4},{4,7},{0,4} };

void link_pix(c,x,y,dir)
{
        int sx, sy;

        if (x<0 || y<0 || x>=X_SIZE || y>=Y_SIZE)
                return;

        SetAPen(rp,c+3);        /* Non usa i colori della
                                        barra del titolo */

        sx = x<<3;
        sy = (y<<3)+10;
        Move(rp,sx+4,sy+4);
        Draw(rp,sx + new_data[dir].x,
sy+new_data[dir].y);
}

void wpix(c,x,y,dir)
{
        int sx, sy;

        if (x<0 || y<0 || x>=X_SIZE || y>=Y_SIZE)
                return;

        scr_image[(x>>3) + y*X_BYTES] |= power[x&7];

        SetAPen(rp,c+3);        /* Non usa i colori della
                                        barra del titolo */
        sx = x<<3;
        sy = (y<<3)+10;
        Move(rp,sx + old_data[dir].x, sy+old_data[dir].y);
        Draw(rp,sx+4,sy+4);
}

flash_colour(c)
{
        UWORD rgb;
        int i,red,grn,blu;
        struct ViewPort *vp;

        vp  = &war_screen->ViewPort;
        rgb = GetRGB4(vp->ColorMap,c);

        red = (rgb >> 8) & 0x0f;
        grn = (rgb >> 4) & 0x0f;
        blu =  rgb       & 0x0f;

        for (i=0;i<10;i++) {
                Delay(5);
                WaitTOF();
                SetRGB4(vp,c,8^red,8^grn,8^blu);
                Delay(5);
                WaitTOF();
                SetRGB4(vp,c,red,grn,blu);
        }
}

draw_scores()
{
        int i;
        char buf[20];
        extern int ties;

        SetBPen(rp,1);
        SetDrMd(rp,JAM2);
        Move(rp,90,7);
        SetAPen(rp,0);
        sprintf(buf,"%-3d ",ties);
        Text(rp,buf,strlen(buf));
        for (i=0;i<MAX_COLOURS;i++) {
                SetAPen(rp,i+3);
                sprintf(buf,"%-3d ",scores[i]);
                Text(rp,buf,strlen(buf));
        }
}
```

Listato 1d

"time.c"

```
/* Funzioni relative al timer.device */


#include <exec/types.h>
#include <exec/ports.h>
#include <devices/timer.h>

#include "cycles.h"

TIMEREQ *Timer_Open(unit,tp)
```

```
ULONG unit;
PORT *tp;
{
        int error;
        TIMEREQ *tr;

        if (!tp) tp = CreatePort(0,0);
        if (!tp) return(0);

        tr = (TIMEREQ *)
                CreateExtIO(tp,sizeof(TIMEREQ));
        if (!tr) {
                printf("Non posso creare una timer
                request\n");
                return(0);
        }

        error = OpenDevice("timer.device",unit,tr,0);
        if (error) {
                Timer_Close(tr);
                return(0);
        }

        return(tr);
}

/* Chiude il timer device */
void Timer_Close(tr)
TIMEREQ *tr;
{
        PORT *tp;

        if (tr) {
                tp = tr-
>tr_node.io_Message.mn_ReplyPort;
                if (tp) DeletePort(tp);
                CloseDevice(tr);
                DeleteExtIO(tr,sizeof(TIMEREQ));
        }
}

/* Chiede al timer device di rispondere al nostro
                messaggio dopo un certo tempo */
void Timer_Post(tr,secs,micro)
TIMEREQ *tr;
{
        tr->tr_node.io_Command      =
                                TR_ADDREQUEST;
        tr->tr_time.tv_secs     = secs;
        tr->tr_time.tv_micro    = micro;
        SendIO(tr);
}
```

Listato 1e

"cycles.h" - Header file usato da tutti i programmi

```
/*
 * Definizione di tutte le strutture e delle costanti
 */

#include <exec/types.h>
#include <exec/memory.h>
#include <exec/nodes.h>
#include <exec/ports.h>
#include <devices/timer.h>
#include <libraries/dos.h>

/* alcune utili abbreviazioni */
typedef struct Node             NODE;
typedef struct MsgPort          PORT;
typedef struct Message          MSG;
typedef struct List             LIST;
typedef struct timerequest      TIMEREQ;

/* Dati che descrivono ogni cycle */
typedef struct {
        NODE  node;             /* Per mantenere le
                                cose ordinate   */
        struct  Order *msg;     /* Posto per i messaggi
                                in attesa   */
        int     status;         /* stato corrente */
        int     x,y;            /* posizione corrente*/
        int     dir;            /* direzione corrente
        int     colour;         /* colore          */
} CYCLE;

typedef struct {
        int     x;              /* dimensione (x)
                        dell'immagine sullo schermo */
        int     y;              /* dimensione (y)
                        dell'immagine sullo schermo */
        BYTE    *buf;           /* puntatore all'area
                                buffer */
} IMAGE;

/* specifica del messaggio che ci si aspetta di ricevere*/
typedef struct Order {
        MSG     msg;            /* struttura di un
                                messaggio Exec  */
        int     order;          /* comandi da eseguire*/
        int     x,y;            /* parametri       */
        int     dir;            /* direzione       */
        int     status;         /* stato   */
        CYCLE *cycle;           /* puntatore alla
                                struttura cycle */
        IMAGE *table;           /* puntatore alla screen
                                table */
} ORDER;

#define STATUS_NEED_NEW 0       /* nessun
                                cambio di direzione
                                specificato     */
#define STATUS_GOT_MOVE 1       /* cambio di
                                direzione specificato */

#define STATUS_DEAD     100     /* un valore
                                superiore indica che è
                                morto */
#define STATUS_LOSER    101     /* vi siete
                                schiantati      */
#define STATUS_WINNER   102     /* siete gli unici
                                rimasti  */
#define STATUS_AWAITING 103     /* aspettate una
                                nuova partita*/
#define STATUS_REMOVED  104     /* non
                                partecipate più*/

#define ORD_DIRECTION   0       /* il cycle
                                cambia direzione alla
                                prossima mossa*/
#define ORD_AWAIT   1           /* risponde quando inizia
                                la prossima partita*/
#define ORD_RETIRE  2           /* Bye Bye, il gioco è
                                finito  */

#define TURN_LEFT(dir)  (((dir)-1)&3)
```

```
#define TURN_RIGHT(dir) (((dir)+1)&3)

#define DIR_UP          0       /* costanti per le
                                   direzioni */
#define DIR_RIGHT       1       /* è garantito che non
                                   cambieranno nelle */
#define DIR_DOWN        2       /* prossime versioni,
                                   quindi si possono fare*/
#define DIR_LEFT        3       /* come aggiungere 1
                                   (mod 4) per girare a
                                   destra */
                                /* e -1 per girare a
                                   sinistra...  */

#define MAX_COLOURS     5       /* massimo
                                   numero di colori e
                                   giocatori  */
#define X_SIZE          40

#define Y_SIZE          23


extern MSG      *GetMsg();
extern PORT     *CreatePort();
extern void     *AllocMem();
extern void     *CreateExtIO();
extern TIMEREQ      *Timer_Open();
extern void     Timer_Close();
extern void     Timer_Post();

extern int      scores[];
```

Listato 2

Questo è il programma a cui i giocatori devono essere linkati. Il "Manager" si prende cura di registrare il giocatore, di visualizzare una finestra con il colore e il punteggio realizzato, nonché di chiudere tutto quando il giocatore si ritira. Quando si usa il "Manager" bisogna solo scrivere le funzioni strategy_init, strategy e strategy_finish: riferirsi al testo per i dettagli.

```
/* manager.c */

#include "cycles.h"
#include <intuition/intuition.h>

#define PER_ROW     3       /* finestre per riga */
#define HEIGHT  50      /* altezza di ogni finestra */

#ifdef AZTEC_C

extern int Enable_Abort;
#endif

extern char player_name[];

static struct Window    *war_window;
static struct RastPort  *rp;

struct IntuitionBase    *IntuitionBase;
struct GfxBase          *GfxBase;

struct Window           *OpenWindow();
void                    *OpenLibrary();

int wins, losses;
int colour;

char *colours[] = { "Rosso", "Blu", "Bianco", "Verde",
                                        "Giallo" };

main()
{
        int status;
#ifdef AZTEC_C
        Enable_Abort = 0;
#endif
        if (Register(&colour)) exit(0);

        if (strategy_init()) {
                Retire();
                exit(0);
        }
        if (g_init((colour%PER_ROW)*(640/
                        PER_ROW),(colour/
                        PER_ROW)*HEIGHT)) {
                strategy_finish();
                Retire();
                exit(0);
        }

        g_show();

        status = STATUS_NEED_NEW;

        for (;;) {
                if (GetMsg(war_window->UserPort)) {
                        strategy_finish();
                        Retire();
                        g_finish();
                        exit(0);
                }

                switch (status) {

                case STATUS_WINNER:
                        wins++;
                        g_show();
                        status = Await();
                        break;

                case STATUS_LOSER:
                        losses++;
                        g_show();
                        status = Await();
                        break;

                case STATUS_NEED_NEW:
                        status = strategy();
                        if (status < 0) {
                                strategy_finish();
                                Retire();
                                g_finish();
                                exit(0);
                        }
                        status = Direction(status);
                        break;

                case STATUS_REMOVED:
                        WaitPort(war_window->User
                                                Port);
                        GetMsg(war_window->User
                                                Port);
```

```
                        strategy_finish();
                        Retire();
                        g_finish();
                        exit(0);
                }
        }
}

/*
 * Qui c'è tutto il codice per il disegno relativo ai
                                giocatori... *
 */

static struct NewWindow newwindow = {
        0,0,                    /* Angolo superiore
                                sinistro */
        640/PER_ROW-2,HEIGHT-1,  /* Larghezza,
                                altezza*/
        0,1,                    /* DetailPen, BlockPen*/
        CLOSEWINDOW,            /* Flag IDCMP */
         WINDOWCLOSE
        |NOCAREREFRESH
        |SMART_REFRESH
        |WINDOWDRAG
        |WINDOWDEPTH
        ,                       /* flag */0,
/* primo gadget */
        0,                      /* checkmark */
        (UBYTE *)player_name,/* titolo */
        0,                      /* lo schermo verrà
                                riempito dopo */
        0,                      /* bitmap */
        0,0,                    /* largezza e altezza
                                minime */
        -1,-1,                  /* larghezza e altezza
                                massime */
        WBENCHSCREEN                    /* tipo */
};

g_init(x,y)
{
        IntuitionBase = OpenLibrary("intuition.library",0);
        if (!IntuitionBase) {
                g_finish();
                printf("Non riesco ad aprire Intuition\n");
                return(1);
        }

        GfxBase =  OpenLibrary("graphics.library",0);
        if (!GfxBase) {
                g_finish();
                printf("Non riesco ad aprire la libreria
                                                grafica\n");
                return(1);
        }

        newwindow.LeftEdge = x;
        newwindow.TopEdge  = y+20;

        war_window = OpenWindow(&newwindow);
        if (!war_window) {
                g_finish();
                printf("Non riesco ad aprire la
                                                finestra\n");
                return(1);
        }
        rp = war_window->RPort;
        SetBPen(rp,0);
        SetAPen(rp,1);
        SetDrMd(rp,JAM2);
        return(0);
}

g_finish()
{
        if (war_window) CloseWindow(war_window);
        if (GfxBase) CloseLibrary(GfxBase);
        if (IntuitionBase) CloseLibrary(IntuitionBase);
}

g_show()
{
        char buf[40];
        int y = 8;
        int x = 8;

        Move(rp,x,y+=9);
        sprintf(buf,"Giocatore %s",colours[colour]);
        Text(rp,buf,strlen(buf));

        Move(rp,x,y+=9);
        sprintf(buf,"  Vinte: %d",wins);
        Text(rp,buf,strlen(buf));

        Move(rp,x,y+=9);
        sprintf(buf,"  Perse: %d",losses);
        Text(rp,buf,strlen(buf));

        Move(rp,x,y+=9);
        sprintf(buf,"Partite: %d",wins+losses);
        Text(rp,buf,strlen(buf));
}
```

Listato 3

"play.c" - routine fornite per facilitare lo sviluppo di programmi

```
 /* Riferirsi al testo per una spiegazione dettagliata
                                delle singole routine*/

#include "cycles.h"
#ifdef AZTEC_C

extern int Enable_Abort;
#endif

PORT *FindPort();

static ORDER order;
static PORT *sPort;
static PORT *cPort;
static BYTE *screen;
static int  x_size,y_size,x_bytes;
static int  colour;

static xdir[] = {0,1,0,-1};  /* su, destra, giù, sinistra */
static ydir[] = {-1,0,1,0};  /* su, destra, giù, sinistra */

/* Manda un messaggio e attende una risposta */
MSG *Send(msg)
MSG *msg;
{
        PutMsg(sPort,msg);
        WaitPort(cPort);
        return(GetMsg(cPort));
}
```

```
/* Registra il giocatore presso il server */
/* resituisce non zero se la registrazione è andata a
                                        buon fine */
/* e indica con quale colore si giocherà */

Register(my_colour)
int *my_colour;
{

        cPort = CreatePort(0,0);
        if (!cPort) {
                printf("Non posso creare la porta\n");
                return(-1);
        }
        order.msg.mn_Node.ln_Type = NT_MESSAGE
        order.msg.mn_ReplyPort          = cPort;
        order.order = ORD_AWAIT;
        order.cycle = 0;

        /* E' importante che la porta non sparisca tra
                        il momento in cui

         * viene trovata e quando gli si spedisce
                        il messaggio.

         * Così si assicura una fine corretta se il
                        server decide di

         * terminare.*/

        Forbid();
        sPort = FindPort("Cycle Server");
        if (!sPort) {
                Permit();
                printf("Non posso trovare la porta
                Cycle Server\n");
                DeletePort(cPort);
                return(-1);
        }
        PutMsg(sPort,&order);
        Permit();

        WaitPort(cPort);
        GetMsg(cPort);

        if (order.status) {
                printf("Il server sta chiudendo. Non si
                accettano più clienti.\n");
                DeletePort(cPort);
                return(-1);
        }

        colour  = order.cycle->colour;
        x_size  = order.table->x;
        y_size  = order.table->y;
        screen  = order.table->buf;
        x_bytes = (x_size>>3)+ !!(x_size & 7);
        *my_colour = colour;

        return(0);
}

void Retire()
{
        /* Controlla se non ci siete già più. Altrimenti
        non dà ORD_RETIRE */
        if (order.status != STATUS_REMOVED) {
           order.order = ORD_RETIRE;
                Send(&order);
        }
        DeletePort(cPort);
}

static int power[] = {1,2,4,8,16,32,64,128 };

/* restituisce lo stato del pixel x,y */
Inquire(x,y)
{
        if (x<0 || y<0 || x>=x_size || y>=y_size) return(1);

        return ((screen[(x>>3)+ y*x_bytes] & power[x&7])
        != 0);
}

/* Restituisce la posizione corrente sullo schermo...
                                        nel caso che il
 * cliente voglia fare un qualche test*/

IMAGE *GetScreenMem()
{
        return(order.table);
}

/* restituisce lo stato dal server, non-zero indica
                                qualcosa di eccezionale,

 * come la dichiarazione di un perdente (LOSER) o di
                                un vincitore (WINNER)

 */


Direction(dir)
{
        order.order = ORD_DIRECTION;
        order.dir = dir;
        Send(&order);
        return(order.status);
}

/* Aspetta che cominci una nuova partita */
Await()
{
        order.order = ORD_AWAIT;
        Send(&order);
        return(order.status);
}

GetInfo(x,y,dir)
int *x, *y, *dir;
{
        *x   = order.x;
        *y   = order.y;
        *dir = order.dir;
}

Look(dir)
int dir;
{
        return (Inquire(order.x + xdir[dir], order.y +
                                        ydir[dir]));
}
```

I giocatori: di seguito sono riportati alcuni esempi di giocatori (player) per fornire un'idea di come si possono

scrivere. I player possono variare da una semplice routine di dieci righe fino a programmi di centinaia di linee di codice.

Giocatore 1

```
/*
        Ecco un esempio di un semplice giocatore.
        Come suggerisce il suo nome (in
        inglese right significa destra), Mr.Right gira a
        destra se la strada è
        bloccata, altrimenti gira a sinistra. Come potete
        vedere scrivere un
        semplice giocatore usando il "manager" è un
        compito molto semplice.
*/

#include "cycles.h"
char player_name[] = "Mr. Right";
strategy_init() {}
strategy_finish() {}
strategy()
{
        int x,y,dir;

        GetInfo(&x,&y,&dir);
        if (!Look(dir))
                return(dir);
        if (!Look(TURN_RIGHT(dir)))
                return(TURN_RIGHT(dir));

        return(TURN_LEFT(dir));
}
```

Giocatore 2

```
/*
        "Dizzy" cerca di girare a sinistra; se non può
        cerca di farlo a destra
        e se non può nuovamente continua sulla
        direzione che stava seguendo.
*/

#include "cycles.h"

char player_name[] = "Dizzy";

strategy_init() {}
strategy_finish() {}

strategy()
{       int x,y,dir;

        GetInfo(&x,&y,&dir);

        if (!Look(TURN_LEFT(dir)))
                return(TURN_LEFT(dir));
        else if (!Look(TURN_RIGHT(dir)))
                return(TURN_RIGHT(dir));
        else
                return(dir);
```

Giocatore 3

```
/*
        Ecco un altro semplice giocatore. "Vertigo" cerca
        di girare se si muove
        orizzontalmente, ma preferisce continuare nella
        direzione presa se è in  senso verticale. In altre
        parole Vertigo preferisce andare su e giù.
*/

#include "cycles.h"
char player_name[] = "Vertigo";
strategy_init() {}
strategy_finish() {}
strategy()
{
        int x,y,dir;
        GetInfo(&x,&y,&dir);
        /* (dir & 1) è vero per sinistra e destra */
        if (dir & 1) {
                if (!Look(TURN_RIGHT(dir)))
                        return(TURN_RIGHT(dir));
                else if (!Look(TURN_LEFT(dir)))
                        return(TURN_LEFT(dir));
                else
                        return(dir);
        }
        else
                if (!Look(dir))
                        return(dir);
                else if (!Look(TURN_RIGHT(dir)))
                        return(TURN_RIGHT(dir));
                else
                        return(TURN_LEFT(dir));
        }
}
```

Giocatore 4

```
/*
        "Round the Clock" è un peso leggero: guarda
        in tutte le celle a sé
        adiacenti e fino a quattro celle lontano.
        Viene calcolato un valore
        per ognuna delle celle in ogni direzione, con
        la cella più vicina
        che ha un valore più alto. Viene poi scelta
        la direzione che ha
        registrato il valore più basso. Una strategia
        come questa può essere
        buona, ma bisogna resistere alla tentazione
        di controllare troppe
        celle avanti a sé, altrimenti si spende
        troppo tempo per decidere
        la prossima mossa.
*/

#include "cycles.h"
char player_name[] = "Round the Clock";
int xdir[] = { 0, 1, 0, -1 };
int ydir[] = { -1, 0, 1, 0 };
strategy_init() {}
strategy_finish() {}
strategy()
{
        int x,y,dir;
      int score,best;
        int i,j;
        GetInfo(&x,&y,&dir);
        best = 1000; /* Più alto del più alto punteggio
possibile */
        for (i=0;i<4;i++) {
                if (Look(i)) continue;
                score = 0;
                for (j=2;j<4;j++)
                        score +=
```

```
Inquire(x+j*xdir[i],y+j*ydir[i]);
                if (score < best ) {
                        best = score;
                        dir = i;
                }
        }
        return(dir);
}
```

Giocatore 5

```
/*
        "Keep'm Flying" usa una strategia simile a
        quella di "Round the Clock",
        eccetto per il fatto che controlla partendo
        dalla direzione originaria.
        In pratica preferisce andare avanti piuttosto
        che girare.
*/

#include "cycles.h"
int xdir[] = { 0, 1, 0, -1 }
int ydir[] = { -1, 0, 1, 0 };
char player_name[] = "Keep'm Flying";
strategy_init() {}
strategy_finish() {}
strategy()
{
        int x,y,dir;
        int score,best;
        int i,j,d;
        GetInfo(&x,&y,&dir);
        best = 8;  /* Più alto del più alto punteggio
                                                possibile */
        for (d=dir,i=0;i<4;i++,d=TURN_RIGHT(d)) {
                if (Look(d)) continue;
                score = 0;
                for (j=2;j<4;j++)
                        score +=
Inquire(x+j*xdir[d],y+j*ydir[d]);
                if (score < best ) {
                        best = score;
                        dir = d;
                }
        }
        return(dir);
}
```

Giocatore 6

```
/*
        "Not so Dizzy" è un peso medio che adotta
        una strategia simile a quella
        dei due giocatori precedenti, usando però
        un algoritmo più sofisticato.
        Notate che, per questioni di velocità, l'area
        di gioco è controllata
        in modo autonomo invece che tramite la
        funzione Inquire(): viene
        mantenuta una copia dell'area di gioco in
        modo che possa essere
        modificata e controllata senza effetti indesiderati.
        Strategy_init e
        strategy_finishservono per allocare e rilasciare
        la memoria.
*/
#include "cycles.h"
char player_name[] = "Not so Dizzy";
int xdir[] = { 0, 1, 0, -1 };
int ydir[] = { -1, 0, 1, 0 };
static IMAGE myTable;
static nbytes;
static IMAGE *table = 0;
static x_bytes;
void mySet(), myClr(), *malloc();
IMAGE *GetScreenMem();
strategy_init()
{
        table = GetScreenMem();
        myTable.x = table->x;
        myTable.y = table->y;
        x_bytes = (myTable.x>>3)+(!!(myTable.x&7));
        nbytes = myTable.y * x_bytes;
        myTable.buf = (BYTE *)malloc(nbytes);
        if (!myTable.buf) {
                printf("Errore, non posso allocare
                la memoria\n");
                return(-1);
        }
        return(0);
}
strategy_finish()
{
        free(myTable.buf);
}
strategy()
{
        int x,y,dir;
        int score,best;
        int i,j,d;

        GetInfo(&x,&y,&dir);
        best = 0;
        copyBoard();
        for (d=TURN_RIGHT(dir),i=
        0;i<4;i++,d=TURN_RIGHT(d)) {
                score = value(x+xdir[d], y+ydir[d], 2);
                if (score >best) {
                        dir = d;
                        best = score;
                }
        }
        return(dir);
}

value(x,y,d)
{
        int i,best,v;
        if (myInquire(x,y)) return(0);
        if (d>0) {
                mySet(x,y);
                best = 0;
                for (i=0;i<4;i++) {
                        v = value(x+xdir[i], y+ydir[i],d-1);
                        if (v>best) best=v;
                }
                myClr(x,y);
                return(1+best);
        }
        return(1);
}
copyBoard()
{
        register i;
        register char *p, *q;
        q = myTable.buf;
        p = table->buf;
```

```
        for (i=nbytes;—i>=0;)
                *q++ = *p++;
}

static power[] = {1,2,4,8,16,32,64,128};
/* resituisce lo stato del pixel x,y */
myInquire(x,y)
{

        if (x<0 || y<0 || x>=myTable.x || y>=myTable.y)
        return(1);
        return ((myTable.buf[(x>>3)+ y*x_bytes] &
        power[x&7]) != 0);
}

void mySet(x,y)
{
        if (x<0 || y<0 || x>=myTable.x || y>=myTable.y)
        return;
        myTable.buf[(x>>3)+ y*x_bytes] |= power[x&7];
}

void myClr(x,y)
{
        if (x<0 || y<0 || x>=myTable.x || y>=myTable.y)
return;
        myTable.buf[(x>>3)+ y*x_bytes] &=
        ~power[x&7];
}
```

Giocatore 7

```
/*
        "Look B4 U Leap" usa una strategia simile a
        "Not So Dizzy" eccetto per il
        fatto che preferisce la direzione corrente.
        Questo giocatore vince quasi
        sempre in tornei giocati con 20 millisecondi.

*/

#include "cycles.h"
char player_name[] = "Look B4 U Leap";
int xdir[] = { 0, 1, 0, -1 };
int ydir[] = { -1, 0, 1, 0 };
static IMAGE myTable;
static nbytes;
static IMAGE *table = 0;
static x_bytes;
void mySet(), myClr(), *malloc();
IMAGE *GetScreenMem();
strategy_init()

{
        table = GetScreenMem();
        myTable.x = table->x;
        myTable.y = table->y;
        x_bytes = (myTable.x>>3)+(!!(myTable.x&7));
        nbytes = myTable.y * x_bytes;
        myTable.buf = (BYTE *)malloc(nbytes);
        if (!myTable.buf) {
                printf("Errore, non posso allocare
                la memoria\n");
                return(-1);
        }
        return(0);
}

strategy_finish()
{
        free(myTable.buf);
}
strategy()

{
        int x,y,dir;
        int score,best;
        int i,j,d;
        GetInfo(&x,&y,&dir);
        best = 0;
        copyBoard();
        for (d=dir,i=0;i<4;i++,d=TURN_RIGHT(d)) {
                score = value(x+xdir[d], y+ydir[d], 3);
                if (score >best) {
                        dir = d;
                        best = score;
                }
        }
        return(dir);
}

value(x,y,d)
{
        int i,best,v;
        if (myInquire(x,y)) return(0);

        if (d>0) {
                mySet(x,y);
                best = 0;
                for (i=0;i<4;i++) {
                        v = value(x+xdir[i], y+ydir[i],d-1);
                        if (v>best) best=v;
                }
                myClr(x,y);
                return(1+best);
        }
        return(1);
}

copyBoard()

{
        register i;
        register char *p, *q;
        q = myTable.buf;
        p = table->buf;
        for (i=nbytes;—i>=0;)
                *q++ = *p++;
}

static power[] = {1,2,4,8,16,32,64,128};
/* restituisce lo stato del pixel x,y */
myInquire(x,y
)
{
        if (x<0 || y<0 || x>=myTable.x || y>=myTable.y)
        return(1);
        return ((myTable.buf[(x>>3)+ y*x_bytes] &
        power[x&7]) != 0);
}
void mySet(x,y)
{
        if (x<0 || y<0 || x>=myTable.x || y>=myTable.y)
        return;
        myTable.buf[(x>>3)+ y*x_bytes] |= power[x&7];
}
```

```
void myClr(x,y)
{
        if (x<0 || y<0 || x>=myTable.x || y>=myTable.y)
return;
        myTable.buf[(x>>3)+ y*x_bytes] &=
~power[x&7];
}
```

Giocatore 8

```
/*
        Questo giocatore non adotta nessuna strategia.
        Si muove semplicemente
        nella direzione indicata da un joystick.
*/

#include "cycles.h"
#include <exec/types.h>
#include <exec/devices.h>
#include <devices/gameport.h>
#include <devices/inputevent.h>
char player_name[] = "Joystick";
static struct MsgPort  *gameport;
static struct IOStdReq *gameio;
static BYTE gamebuffer[sizeof(struct InputEvent)];
static BYTE *gamebuff;
static struct InputEvent *gamedata;
struct IOStdReq *CreateStdIO();
strategy_init()

{
        gameport = CreatePort(0,0);
        if (!gameport) {
                printf("Errore, non posso creare
        la porta\n");
                return(-1);
        }

        gameio = CreateStdIO(gameport);
        if (!gameio) {
                printf("Errore, non posso creare
        il messagio StdIO\n");
                DeletePort(gameport);
                return(-1);
        }

        if (OpenDevice("gameport.device",1,gameio,0)) {
                printf("Errore, non posso aprire
        il  gameport.device\n");
                DeleteStdIO(gameio);
                DeletePort(gameport);
                return(-1);
        }

        if (SetCType(GPCT_ABSJOYSTICK)) {
                printf("Errore, non posso decidere il
        tipo di controller\n");
                CloseDevice(gameio);
                DeleteStdIO(gameio);
                DeletePort(gameport);
                return(-1);
        }

        if
(SetTrigger(GPTF_UPKEYS|GPTF_DOWNKEYS,50)) {
                printf("Errore, non posso impostare il
                tipo di trigger\n");
                CloseDevice(gameio);
                DeleteStdIO(gameio);
                DeletePort(gameport);
                return(-1);
        }

        gameio->io_Command = GPD_READEVENT;
        gameio->io_Data          = (APTR)gamebuffer;
        gameio->io_Length  = sizeof(struct InputEvent);
        gameio->io_Flags   = 0;
        gamedata = (struct InputEvent *)gamebuffer;
}
strategy_finish()

{
        CloseDevice(gameio);
        DeleteStdIO(gameio);
        DeletePort(gameport);
}

strategy()

{
        int code;
        int xmove, ymove;
        int x,y,dir;
        GetInfo(&x, &y, &dir);
        gameio->io_Length  = sizeof(struct InputEvent);
        SendIO(gameio);
        WaitPort(gameport);
        while (GetMsg(gameport)) ;
        code  = gamedata->ie_Code;
        xmove = gamedata->ie_X;
        ymove = gamedata->ie_Y;
        if (ymove)
                return( (ymove<0) ? DIR_UP
                :DIR_DOWN );
        if (xmove)
                return( (xmove<0) ? DIR_LEFT
                :DIR_RIGHT );
        return(dir);
}

static SetCType(type)

{
        gameio->io_Command = GPD_SETCTYPE;
        gameio->io_Length  = 1;
        gameio->io_Data    = (APTR)gamebuff;
        *gamebuff          = type;
        SendIO(gameio);
        WaitPort(gameport);
        GetMsg(gameport);
        return(gameio->io_Error);
}

static SetTrigger(keys,timeout)

{
        struct GamePortTrigger gpt;
        gameio->io_Command = GPD_SETTRIGGER;
        gameio->io_Length  = sizeof(gpt);
        gameio->io_Data          = (APTR)&gpt;

        gpt.gpt_Keys       = keys;
        gpt.gpt_Timeout          = timeout;
        gpt.gpt_XDelta     = 1;
        gpt.gpt_YDelta     = 1;
        return(DoIO(gameio));
}
```

LISTINO LIBRI E GRANDI OPERE JACKSON

CODICE	TITOLO	PREZZO	
	INFORMATICA: CONCETTI GENERALI		
511 A	COME PROGRAMMARE	15.000	
503 A	PROGRAMMAZIONE STRUTTURATA, CORSO DI AUTOISTRUZIONE	15.000	
101 H	TERMINI DELL'INFORMATICA E DELLE DISCIPLINE CONNESSE	50.000	
539 A	LOGICA E DIAGRAMMI A BLOCCHI: TECNICHE DI PROGRAMMAZIONE	40.000	
526 P	DATA BASE: CONCETTI E DISEGNO	22.500	
GYS190	TRADUTTORI DI LINGUAGGI	26.000	
G 240	PAROLE BASE DELL'INFORMATICA	8.000	
GYS245	CONCETTI DI INFORMATICA	43.000	
GYS248	DATA PROCESSING	45.000	
GY 264	DATA FILE	50.000	
GYS266	ARCHITETTURE DI SISTEMA	32.000	
GY 354	SISTEMI INTELLIGENTI	28.000	
CZ 419	ANALISI E PROGRAMMAZIONE	11.000	
158 EC	INFORMATICA DI BASE I CONCETTI FONDAMENTALI HARDWARE E SOFTWARE	55.000	
526 A	VOI E L'INFORMATICA	15.000	
100 H	DIZIONARIO DI INFORMATICA	59.000	
GY 551	I LINGUAGGI DELLA 4a GENERAZIONE	65.000	
GYS552	PRIMA DEL LINGUAGGIO LA PROGRAMMAZIONE	35.000	
GYS 559	C.S.P. - PROCESSI SEQUENZIALI	49.000	
GYS 546	ALGORITMI FONDAMENTALI	54.000	
GY 618	SISTEMI ESPERTI	28.000	
047 T	MICROPROCESSORI	14.500	
048 T	DATA BASE	14.500	
049 T	FILE	14.500	
CI 686	CAPIRE IL PERSONAL COMPUTER	35.000	
G 540	MODELLI MATEMATICI E SIMULAZIONE	56.000	
GE 688	ENCICLOPEDIA MONOGRAFICA DI ELETTR. E INF. VOLUME I	58.000	
GE 689	ENCICLOPEDIA MONOGRAFICA DI ELETTR. E INF. VOLUME II	58.000	
GY 629	SOFTWARE DI BASE - Strumenti di sviluppo	52.000	
	INFORMATICA: SISTEMI OPERATIVI		
G 223	UNIX LA GRANDE GUIDA	70.000	
GY 272	SISTEMI OPERATIVI PER MICROCOMPUTER	25.000	
GY 273	MS-DOS LA GRANDE GUIDA	45.000	
510 P	CP/M CON MP/M	29.000	
CZ 538	MS DOS 2 E 3	49.000	
G 543	XENIX	45.000	
R 588	LAVORARE CON XENIX	70.000	
GYS271	SISTEMI OPERATIVI	55.000	
R 615	I COMANDI DI XENIX MAIL	12.500	
092 D	SOFTWARE DI BASE E SISTEMI OPERATIVI	7.000	
093 D	CP/M IL "SOFTWARE BUS"	7.000	
094 D	MS-DOS E PC-DOS LO STANDARD IBM	7.000	
009 H	UNIX	8.500	
011 H	CP/M	8.500	
044 T	MS DOS	14.500	
045 T	PC DOS	14.500	
R 628	MICROSOFT OS/2	50.000	
046 T	UNIX	14.500	
MS 02 E	COFANETTO "MS-DOS" 5"¼ - Corso autoistruzione	156.000	9F
R 600	MS DOS ADVANCED - Il Manuale del Programmatore	55.000	
GY 663	UNIX PROGRAMMAZIONE AVANZATA	55.000	
BY 724	GUIDA AI SISTEMI OPERATIVI	29.000	
BY 744	UNIX: CONCETTI, STRUTTURE, UTILIZZO	43.000	
R 761	MS-DOS REFERENCE GUIDE	14.500	
	INFORMATICA: LINGUAGGI		
501 A	IMPARIAMO IL PASCAL	16.000	
502 A	INTRODUZIONE AL BASIC	25.000	
500 P	PASCAL MANUALE E STANDARD DEL LINGUAGGIO	16.000	
329 A	PROGRAMMARE IN ASSEMBLER	14.000	
513 A	PROGRAMMARE IN BASIC	8.000	
514 A	PROGRAMMARE IN PASCAL	19.000	
516 A	INTRODUZIONE AL PASCAL	39.000	

CODICE	TITOLO	PREZZO	
517 P	DAL FORTRAN IV AL FORTRAN 77 (II ED.)	32.000	
521 A	50 ESERCIZI IN BASIC	17.000	
525 A	BASIC PER TUTTI	23.000	
534 A	MANUALE DEL BASIC	45.000	
509 A	LOGO: POTENZA E SEMPLICITÀ	20.500	
507 B	TUO PRIMO PROGRAMMA IN BASIC (II)	19.500	
533 A	BASIC DALLA A ALLA Z	19.000	
540 A	LINGUAGGIO ADA	19.500	
541 P	LINGUAGGIO C	25.000	
542 P	COBOL STRUTTURATO: CORSO DI AUTOISTRUZIONE	50.000	
508 P	PROGRAMMARE IN C	39.000	
G 233	COBOL PER MICROCOMPUTER	35.000	
GYS246	ESERCIZI DI FORTRAN	20.000	
GYS247	ESERCIZI IN PASCAL: ANALISI DEI PROBLEMI	29.000	
GYS254	PROGRAMMAZIONE IN LINGUAGGIO ADA	42.000	
GY 270	APL PER IL P.C. IBM	25.000	
GYS274	DAL PASCAL AL MODULA 2	26.000	
GYS311	LINGUAGGIO C IL LIBRO DELLE SOLUZIONI	24.000	
GYS328	APPLICAZIONI IN PASCAL	32.000	
GY 535	TURBO PASCAL	29.000	
G 544	"C" LIBRARY	49.000	
GYS550	PROLOG - LINGUAGGIO E APPLICAZIONE	32.000	
R 589	TURBOPASCAL - LIBRERIA DI PROGRAMMI	45.000	
042 T	LINGUAGGIO C	12.500	
108 D	FORTH ANATOMIA DI UN LINGUAGGIO	7.000	
107 D	FORTRAN E COBOL LINGUAGGI SEMPRE VERDI	7.000	
086 D	ED È SUBITO BASIC VOL. 1	7.000	
087 D	ED È SUBITO BASIC VOL. 2	7.000	
034 T	PROLOG	14.000	
035 T	LISP	12.500	
001 H	COBOL	8.500	
006 H	PASCAL	8.500	
007 H	BASIC	8.500	
010 H	FORTRAN 77	8.500	
020 H	LOGO	8.500	
022 H	FORTH	8.500	
R 612	TURBO PROLOG	50.000	
GY 626	IL MANUALE DEL PASCAL	42.000	
GY 616	DEBUGGING C	55.000	
GY 687	DALLA PROGRAMMAZIONE STRUTTURATA AL PASCAL	42.000	
GY 634	FONDAMENTI DI COMMON LISP	40.000	
	INFORMATICA: LAVORO È SOCIETÀ		
519 P	COMPUTER GRAFICA	29.000	
800 P	ODISSEA INFORMATICA	50.000	
407 H	APPLICAZIONI DEL COMPUTER NELL'UFFICIO MODERNO	23.000	
802 H	INFORMATICA MUSICALE	27.000	
802 P	COMPUTERGRAPHIA	40.000	
806 P	COMPUTER PER L'INGEGNERIA EDILE	22.000	
807 P	COMPUTER PER IL MEDICO	19.000	
CI 231	COMPUTER IMAGE	40.000	
CI 241	ODISSEA INFORMATICA STRATEGIE CULTURALI PER UNA SOCIETÀ INF.	32.000	
G 400	COMPUTER GRAPHICS E ARCHITETTURA	27.000	
PV 409	COMPUTER GRAPHICS E MEDICINA	18.000	
GY 487	MEDICO & COMPUTER	45.000	
GY 548	INFORMATICA MEDICA	65.000	
PA 685	OFFICE AUTOMATION	28.000	
RA 596	DESKTOP PUBLISHING	35.000	
050 T	WORD	14.500	
	INFORMATICA: SOFTWARE PACCHETTI APPLICATIVI		
570 P	CONTABILITÀ COL PERSONAL COMPUTER	27.000	
525 P	WORDSTAR	24.000	
546 P	MANUALE DEL DBASE II	24.000	
578 P	PC NELL'ORG. DELLE PICCOLE AZIENDE: APPL. DEL MULTIPLAN	29.000	F
PP 219	LOTUS 1-2-3: GUIDA ITALIANA ALL'USO	21.000	

CODICE	TITOLO	PREZZO	
G 234	RIORDINO E GESTIONE DEGLI ARCHIVI APPLICAZIONI CON PFS-FILE	30.000	
PP 255	DBASE III GUIDA ITALIANA ALL'USO	45.000	
PA 282	MODELLI DECISIONALI PER IL MANAGER	50.000	
PA 288	PIANIFICAZIONE AZIENDALE PLANNING, MARKETING STRAT., BUDGETING	35.000	F
PP 310	LA GRANDE GUIDA LOTUS A SYMPHONY	70.000	
PP 326	MULTIPLAN CORSO D'ISTRUZIONE	40.000	F
PP 344	FRAMEWORK II - GUIDA ITALIANA ALL'USO	27.000	
PP 351	WORD PROCESSING	27.000	
PP 467	IMPARA 1-2-3 CON LA GRANDE GUIDA LOTUS	45.000	
PP 468	CHART - CORSO ISTRUZIONE	45.000	F
PP 473	IL NUOVO 1-2-3 GUIDA ALL'USO DELLA VERSIONE ITALIANA 2 LOTUS 1-2-3	29.000	
PA 474	BILANCIO, BUDGET, CASH FLOW	40.000	F
PP 475	DBASE III - CORSO DI PROGRAMMAZIONE	23.000	
PA 476	PREVISIONE, PIANIFICAZIONE, SIMULAZIONE CON LOTUS 1-2-3	60.000	F
PV 477	GUIDA ALLA BUSINESS GRAPHIC	20.000	
PP 480	AUTOCAD	40.000	
PP 481	RBASE 5000 - GUIDA ITALIANA ALL'USO	20.000	
PP 537	IL MANUALE DI WINDOWS	60.000	
PP 539	DBASE III - TECNICHE AVANZATE DI PROGRAMMAZIONE	42.000	
PP 545	APPLICAZIONI DI DBASE III	50.000	F
PA 566	MODELLI DECISIONALI CON LOTUS 1-2-3	40.000	F
PP 577	MANUALE DBASE III PLUS	49.000	
039 T	WORDSTAR	12.500	
040 T	LOTUS 1-2-3	12.500	
043 T	WINDOWS	12.500	
PP 621	I COMANDI DI DBASE III PLUS	12.500	
095 D	GUIDA AI PACKAGE APPLICATIVI MERCEOLOGIA DEL SOFTWARE	7.000	
096 D	VISICALC GUIDA RAPIDA ALL'UTILIZZO	7.000	
098 D	WORD PROCESSING	7.000	
103 D	LOTUS 1-2-3 E SIMPHONY IL FASCINO DELL'INTEGRAZIONE	7.000	
104 D	DBASE II E III I PRINCIPI DI DATABASE	7.000	
106 D	MULTIPLAN SPREADSHEET MULTISTRATO	7.000	
110 D	PACKAGE A CONFRONTO PROVE DEI SOFTWARE PIÙ DIFFUSI	7.000	
031 T	FRAMEWORK E FRAMEWORK II	12.500	
033 T	MULTIPLAN 2.02	12.500	
036 T	SYMPHONY	12.500	
038 T	REFLEX	12.500	
027 H	EASY SCRIPT	8.500	
033 H	PAGE MAKER	8.500	
034 H	PROJECT	8.500	
035 H	RBASE	8.500	
PP 611	GUIDA ALL'USO PROFESSIONALE REFLEX	55.000	
PP 636	MANUALE DI WORD	70.000	
PP 594	GUIDA ALL'USO PROFESSIONALE DI LOTUS 1-2-3	50.000	
PP 593	VENTURA - Il grande manuale	55.000	
R 671	LINGUAGGIO C - Reference guide	12.500	
051 T	I COMANDI DI LOTUS 1-2-3 - Reference guide	12.500	
PP 581	PROGRAMMARE IN FRED	40.000	
PP 631	dBASE III PLUS - Guida uso professionale	65.000	
PP 694	PROGRAMMARE IN WINDOWS	70.000	
PA 592	GESTIONE DELLA PRODUZIONE	40.000	F
PP 727	VENTURA - REFERENCE GUIDE	14.500	
PP 700	MATEMATICA CON LOTUS 1-2-3	35.000	F
R 574	MANUALE DELLE STAMPANTI LASER	25.000	
PP 641	AUTOCAD - Il grande manuale	55.000	
PP 728	VENTURA - Fogli stile	42.000	F
PP 741	WORD 3 e 4	59.000	F
PP 642	AUTOCAD Programmazione avanzata	65.000	F
BY 707	ORACLE	75.000	
PA 771	MODELLI PER LOTUS 1-2-3	28.000	F
	PERSONAL COMPUTER		
550 D	PROGRAMMI PRATICI IN BASIC	15.000	
515 H	BASIC E LA GESTIONE DEI FILE VOL. I: METODI PRATICI	15.000	

CODICE	TITOLO	PREZZO	
551 D	75 PROGRAMMI IN BASIC PER IL VOSTRO COMPUTER	12.000	
552 D	PROGRAMMI DI MATEMATICA E STATISTICA IN BASIC	20.000	
554 P	PROGRAMMI SCIENTIFICI IN PASCAL	29.000	
516 H	BASIC E LA GESTIONE DEI FILE - VOL. 2	17.000	
CH 182	COMPUTER HARDWARE REALIZZ. PRATICHE PER GLI HC PIÙ DIFFUSI	18.000	
CI 187	COMPUTER L'HOBBY E IL LAVORO	12.000	
G 235	GRAFICA PER PERSONAL COMPUTER	39.000	
GE 263	METODI DI INTERFACC. PERIFERICHE	43.000	
GE 402	CORSO DI AUTOISTRUZIONE PER MICROCOMPUTER VOL. 1 + VOL. 2	35.000	
PA 406	COME GESTIRE LA PICCOLA AZIENDA CON IL P.C.	22.000	
PP 408	BUSINESS IN BASIC	23.000	
CI 412	IL COMPUTER È UNA COSA SEMPLICE	15.000	
CC 415	CONTROLLO DEI DISPOSITIVI DOMESTICI CON IL P.C.	23.000	
159 GC	PERSONAL COMPUTER DAL SOFTWARE DI BASE ALLE APPLICAZIONI D'UFFICIO	55.000	
R 587	HARD DISK - LA GRANDE GUIDA	75.000	
084 D	INTRODUZIONE AI PERSONAL COMPUTER VIVERE COL PC	7.000	
099 D	SCRIVERE UN'AVVENTURA, 1000 AVVENTURE COL PROPRIO PC	7.000	
100 D	GRAFICA E BASIC LE BASI DELLA COMPUTERGRAFICA	7.000	
085 D	HARDWARE DI UN PERSONAL COMPUTER DENTRO E FUORI LA SCATOLA	7.000	
101 D	GESTIONE DEI FILE IN BASIC E PASCAL VOL. 1	7.000	
102 D	GESTIONE DEI FILE IN BASIC E PASCAL VOL. 2	7.000	
113 D	DISEGNARE COL PERSONAL COMPUTER	7.000	
105 D	PERSONAL E HOME COMPUTER A CONFRONTO	7.000	
112 D	SUONO E MUSICA COL PERSONAL COMPUTER	7.000	
109 D	COSTRUIRSI UN PERSONAL DATABASE	7.000	
097 D	GUIDA ALL'ACQUISTO DI UN PERSONAL COMPUTER	7.000	
088 D	TO DO OR NOT TO DO COME AVER CURA DEL PROPRIO PC	7.000	
089 D	SOFTWARE STRUTTURATO CON ELEMENTI DI PASCAL	7.000	
090 D	DIZIONARIO DI INFORMATICA	7.000	
091 D	BASI DELLA PROGRAMMAZIONE STENDERE UN PROG. COME SI DEVE	7.000	
004 H	PROGRAMMAZIONE	8.500	
015 H	PROGRAMMI DI STATISTICA	8.500	
	PERSONAL COMPUTER: COMMODORE		
347 D	VOI E IL VOSTRO COMMODORE 64	24.000	
348 D	COMMODORE 64 - IL BASIC	28.000	
400 D	FACILE GUIDA AL COMMODORE 64	13.500	
400 B	COMMODORE 64 - FILE	19.000	
409 B	COMMODORE 64 - LA GRAFICA E IL SUONO	34.000	C
570 D	MATEMATICA E COMMODORE 64	26.500	C
350 D	LIBRO DEI GIOCHI DEL COMMODORE 64	24.000	C
575 D	TECNICHE DI PROGRAMMAZIONE SUL COMMODORE 64	16.500	
572 D	LINGUAGGIO MACCHINA DEL COMMODORE 64	35.000	F
576 D	SISTEMA TOTOMAC: LA NUOVA FRONTIERA DEL TOTOCALCIO	29.000	C
548 B	64 PERSONAL COMPUTER E C64	45.000	
SDP222	STATISTICA AD UNA DIMENSIONE CON IL C64	24.000	
CC 260	AVVENTURE (COMMODORE 64)	20.000	C
CC 320	AMIGA HANDBOOK	35.000	
CC 322	COMMODORE 128 OLTRE IL MANUALE	29.000	
CC 323	PROGRAMMI PER COMMODORE 128	29.000	C
CZ 541	128 E 64 - LE PERIFERICHE	32.000	
CC 564	MANUALE RIPARAZIONE C64	55.000	
CZ 532	MANUALE DI AMIGA	39.000	
002 H	COMMODORE 64	8.500	
CC 658	GRAFICA E SUONO PER C64 - 64PC - C128	35.000	F
CC 657	MANUALE DEL COMMODORE C64 - C64PC - C128	35.000	F
CC 627	AMIGA 500	55.000	
CC 750	C.128 LA GRANDE GUIDA	50.000	
CC 749	C.64 LA GRANDE GUIDA	50.000	
	PETER NORTON		
R 734	MANUALE DEL DOS	55.000	
R 736	INSIDE PC IBM	63.000	
R 733	HARD DISK COMPANION	60.000	
R 735	LINGUAGGIO ASSEMBLY PER PC IBM	72.000	2F

CODICE	TITOLO	PREZZO	
	PERSONAL COMPUTER: IBM		
564 D	PROGRAMMI UTILI PER IBM PC	19.000	
G 217	GRAFICA PER IL PERSONAL COMPUTER IBM	39.000	
GY 319	PC IBM MANUALE DEL LINGUAGGIO MACCHINA	45.000	
GY 335	MAPPING PC IBM GESTIONE DELLA MEMORIA	42.000	
PP 407	MANUALE BASE DEL PC IBM	22.000	
041 T	PC IBM	12.500	
R 609	SOLUZIONI AVANZATE PER IL PROGRAMMATORE	60.000	
CZ 751	AVVENTURE PER MS-DOS	35.000	F
RA 484	GUIDA ALLE RETI DI PC IBM	46.000	
	PERSONAL COMPUTER: OLIVETTI		
401 P	PRIMO LIBRO PER M24: MS DOS E GW BASIC	28.000	
401 B	OLIVETTI M10: GUIDA ALL'USO	18.000	
CL 216	BASIC IN 30 ORE PER M24 ED M20	32.000	
CZ 483	MANUALE OLIVETTI M19	42.000	
CZ 536	MANUALE PC 128 OLIVETTI PRODEST	29.000	
CZ 582	PROGR. PER PC 128 OLIVETTI PRODEST (CASS.)	27.000	
	PERSONAL COMPUTER: MSX		
CZ 181	30 PROGRAMMI PER MSX	20.000	C
417 D	MSX: IL BASIC	23.000	
CC 261	AVVENTURE (MSX)	20.000	C
CC 289	SUPER PROGRAMMI PER MSX	35.000	C
CC 336	MSX LA GRAFICA	25.000	
111 D	STANDARD MSX	7.000	
	PERSONAL COMPUTER: APPLE		
331 P	APPLE II GUIDA ALL'USO	31.000	
416 P	MACINTOSH NEGLI AFFARI: MULTIPLAN E CHART	16.500	
424 P	UN MAC PER AMICO: USO, APPLICAZIONI E PROGRAMMI PER MACINTOSH	12.000	
PP 224	MACINTOSH ARTISTA: MACPAINT E MACDRAW	16.000	
CCP277	APPLE IIC GUIDA ALL'USO	45.000	
CC 312	PROGRAMMI PER APPLE IIC	13.000	
CC 417	PROGRAMMI COMM. E FINANZIARI CON APPLE	22.000	
340 H	APPLE MEMO	15.000	
CC 576	IL MANUALE DELL'APPLE II GS	28.000	
003 H	APPLE IIE IIC	8.500	
CC 665	MICROSOFT BASIC PER APPLE MACINTOSH	32.000	F
	PERSONAL COMPUTER: ATARI - AMSTRAD - SHARP		
540 H	BASIC ATARI	18.000	
CC 330	PROGRAMMI PER AMSTRAD CPC 464 CPC 664 - CPC 6128	29.000	C
CC 331	PROGRAMMI PER ATARI 130XE	19.000	C
CC 471	MANUALE ATARI 520 ST E 1040 ST	28.000	
CC 486	WORD PROCESSING CON AMSTRAD PCW 8256/8512	35.000	
032 T	AMSTRAD PCW 8256 e PCW 8512	14.000	
028 H	AMSTRAD 464 E 664	8.500	
	COMMUNICATION E TELEMATICA		
309 A	PRINCIPI E TECNICHE DI ELABORAZIONE DATI	20.000	
518 D	TELEMATICA	28.000	
528 P	TRASMISSIONE DATI	27.000	
617 P	RETI DATI: CARATTERISTICHE, PROGETTO E SERVIZI TELEMATICI	40.000	
GYS314	ELABORAZIONE DIGITALE DEI SEGNALI: TEORIA E PRATICA	25.000	
PA 327	BANCHE DATI RICERCA ONLINE	26.000	
158 LC	COMUNICAZIONI DALLE ONDE ELETTROMAGNETICHE ALLA TELEMATICA	55.000	
CC 472	MODEM E PC USO E APPLICAZIONI	25.000	
GTS478	RETI LOCALI	44.000	
GTS479	IL MODEM - TEORIA, FUNZIONAMENTO	28.000	
R 542	TRASMISSIONE DATI PER PC	31.000	
GT 555	LA TELEMATICA NELL'UFFICIO	35.000	
R 601	COLLEGAMENTO TRA MICRO E MAINFRAME	39.000	
BT 655	MANUALE DI TV E VIDEO COMMUNICATION	45.000	

CODICE	TITOLO	PREZZO
	ELETTRONICA DI BASE E TECNOLOGIA	
201 A	CORSO DI ELETTRONICA FONDAMENTALE CON ESPERIMENTI	35.000
204 A	ELETTRONICA INTEGRATA DIGITALE	50.000
205 A	MANUALE PRATICO DI PROGETTAZIONE ELETTRONICA	35.000
200 A	SISTEMI DIGITALI: MANUTENZIONE, RICERCA ED ELIMINAZIONE GUASTI	28.500
GES262	TECNOLOGIE VLSI	70.000
GES390	ELETTRONICA INTEGRATA DIGITALE IL LIBRO DELLE SOLUZIONI	17.000
CE 411	LA FISICA DEI SEMICONDUTTORI	10.000
158 PC	ELETTRONICA DI BASE I FONDAMENTI DELL'ELETTRONICA ANALOGICA	55.000
158 CC	ELETTRONICA DIGITALE VOL. 1 DALLE PORTE LOGICHE AI CIRCUITI INTEGRATI	55.000
158 DC	ELETTRONICA DIGITALE VOL. 2 DAI BUS AI GATE ARRAY	55.000
158 GC	ELETTROTECNICA ELETTROSTATICA ELETTROMAGNETISMO RETI ELETTR.	55.000
	ELETTRONICA APPLICATA	
601 B	TIMER 555	10.000
203 A	INTRODUZIONE AI CIRCUITI INTEGRATI DIGITALI	10.000
612 P	MANUALE DEGLI SCR VOL. 1	28.000
613 P	MANUALE DI OPTOELETTRONICA	15.000
614 A	FIBRE OTTICHE	15.000
GE 403	JFET MOS E DATA BOOK	20.000
GE 404	TRANSISTOR DATA BOOK	32.000
GE 405	METODI DI PROTEZIONE CONTRO LE SOVRATENSIONI	17.000
CE 413	IL MANUALE DEGLI SCR E TRIAC	15.000
CE 421	MANUALE DEI FILTRI ATTIVI	29.000
CE 423	MANUALE DEI PLL PROGETTAZIONE DEI CIRCUITI	29.000
CE 425	MANUALE DEGLI AMPLIFICATORI OPERAZIONALI	29.000
CE 429	250 PROGETTI CON GLI AMPLIFICATORI DI NORTON	39.000
CE 431	MANUALE DEI CMOS	25.000
CE 485	IL COLLAUDO DELLE SCHEDE	18.000
BE 557	I TRASDUTTORI	43.000
BT 585	FIBRE OTTICHE	29.000
BE 578	MANUALE DI ELETTRONICA	29.000
BE 558	IL MANUALE DEL TECNICO ELETTRONICO	51.000
BE 610	GUIDA ALLA STRUMENTAZIONE ELETTRONICA	34.000
BE 619	MULTIMETRI DIGITALI	42.000
BE 639	ENCICLOPEDIA DEI CIRCUITI INTEGRATI	60.000
BE 654	MANUALE DI ELETTRONICA DEL COMPUTER	20.000
701 P	MANUALE PRATICO DEL RIPARATORE RADIO TV	29.000
705 P	IMPIEGO PRATICO DELL'OSCILLOSCOPIO	17.500
615 P	PROGETTAZIONE DI SISTEMI DI ALTOPARLANTI	21.000
CE 427	L'ELETTRONICA A STATO SOLIDO	25.000
BE 718	77 SCHEDE PER IL RIPARATORE TV	40.000
BE 723	MISURE DEI CIRCUITI ELETTRONICI	26.000
BE 721	MANUALE PRATICO DI ELETTRONICA DIGITALE	26.000
BE 684	IL MANUALE DEI CMOS	35.000
BE 731	IL MANUALE DEGLI AMPLIFICATORI OPERAZIONALI	39.000
	ELETTRONICA: MICROPROCESSORI	
310 P	NANOBOOK Z80 VOL. 1	20.000
007 A	BUGBOOK VII	17.000
314 P	TECNICHE DI INTERFACCIAMENTO DEI MICROPROCESSORI	31.000
312 P	NANOBOOK Z80 VOL. III	25.000
320 P	MICROPROCESSORI DAI CHIPS AI SISTEMI	29.000
324 P	PROGRAMMAZIONE DELLO Z80 E PROGETTAZIONE LOGICA	21.500
326 P	Z80 PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY	50.000
328 D	PROGRAMMAZIONE DELLO Z80	40.000
504 B	APPLICAZIONI DEL 6502	17.000
503 B	PROGRAMMAZIONE DEL 6502	35.000
505 B	GIOCHI CON IL 6502	19.500
G 220	8086-8088 PROGRAMMAZIONE	40.000
GY 265	ASSEMBLER PER IL 68000	70.000
CE 410	IMPIEGO DELLO Z80	23.000
158 HC	MICROPROCESSORI ARCHIT. PROGR. E INTERFAC. DEI MP DA 4 A 32 BIT	55.000

CODICE	TITOLO	PREZZO	
013 H	ASSEMBLER 6502	8.500	
016 H	ASSEMBLER Z80	8.500	
021 H	ASSEMBLER 68000	8.500	
025 H	ASSEMBLER 8086-8088	8.500	
029 H	ASSEMBLER 80286	8.500	
GE 567	80286 ARCHITETTURA E PROGRAMMAZIONE	58.000	
GY 603	80386 ARCHITETTURA E PROGRAMMAZIONE	37.000	
	AUTOMAZIONE		
208 A	CONTROLLORI PROGRAMMABILI	24.000	
616 P	CONTROLLO AUTOMATICO DEI SISTEMI	29.500	
GES251	STRUTTURA E FUNZIONAMENTO DEI CONTROLLI NUMERICI	29.000	
GES252	CONTROLLI NUMERICI: PROGRAMMAZIONE E APPLICAZIONI	28.000	
G 399	30 APPLICAZIONI DI CAD	29.000	
G 401	CAD/CAM & ROBOTICA	28.000	
CI 414	DAL CHIP ALLA ROBOTICA	15.000	
GE 547	LA PROGETTAZIONE AUTOMATICA	32.000	
GE 564	ROBOTICA - Fondamenti e applicazioni	38.000	
	DIZIONARI ENCICLOPEDICI		
DS 498	FISICA	14.000	
DS 499	MATEMATICA	14.000	
DS 522	GEOLOGIA	14.000	
DS 524	ELETTRONICA	14.000	
DS 525	ASTRONOMIA	14.000	
DS 526	CHIMICA	14.000	
DS 527	RAGIONERIA GENERALE	14.000	
DS 528	RAGIONERIA APPLICATA	14.000	
DS 529	BIOLOGIA	14.000	
DS 530	MECCANICA	14.000	
DS 531	INFORMATICA	14.000	
	ARGOMENTI VARI		
704 D	MANUALE PRATICO DI REGISTRAZIONE	10.000	
706 A	COMUNICAZIONI RADIO IN MARE	18.000	
800 H	FENDER, STORIA DI UN MITO	28.000	
CZ 748	TOTOCALCIO, ENALOTTO, TOTIP	35.000	F
	SOFTWARE E MANAGEMENT TOOLS		
CZ 469	GRAFIX - DISEGNARE CON IL PC	50.000	F
TP 606	CORSO AUTOISTRUZIONE LOTUS 1-2-3 (VERS. ITALIANA) F - MS DOS	90.000	F
TY 605	CORSO AUTOISTRUZIONE SUL SISTEMA MS DOS - FLOPPY	50.000	F
TY 640	TURBO PASCAL - LIBRERIA DI PROGRAMMI	40.000	F

CODICE	TITOLO	PREZZO	
TP 643	CORSO AUTOISTRUZIONE LOTUS 1-2-3 (INGLESE)	90.000	F
TP 608	BUDGET STRATEGICO (LOTUS 1-2-3)	100.000	F
TP 614	GESTIONE DELLE COMMESSE DI PRODUZIONE	100.000	F
TP 623	CONTROLLO DELLE VENDITE (CON MULTIPLAN)	100.000	F
TP 625	GESTIONE DEL PERSONALE (LOTUS 1-2-3)	100.000	F
TP 677	GESTIONE DELLE COMMESSE CON MULTIPLAN 2.0	100.000	F
TP 673	PREVENTIVO E CONSUNTIVO DEI COSTI - CON LOTUS 1-2-3 VERS. 2 E MULTIPLAN 2.0	100.000	F
TP 660	1-2-3 LIBRERIA DI MACRO	60.000	F
TY 691	SUPER SCREEN - UTILITY PER I PROGRAMMATORI	50.000	F
TY 690	PC DOCTOR UTILITY - RECOVERING DEI FILE	60.000	F
TP 644	STATISTICA A UNA E DUE DIMENSIONI	100.000	F
TP 681	ANALISI ABC CON LOTUS 1-2-3	100.000	F
TP 669	GESTIONE DELLE COMMESSE CON dBASE III PLUS	100.000	F
	MARKETING & MANAGEMENT		
M 648	PROBLEMI DI MARKETING	45.000	
M 649	DISTINTA BASE	23.000	
M 650	TECNICHE DI ANALISI FINANZIARIA	52.000	
M 647	RICERCHE DI MERCATO	72.000	
	NOVITÀ NOVEMBRE '88		
BE 737	IL MANUALE DEI FILTRI ATTIVI	34.000	
R 730	TURBO C	62.000	F
GY 662	UNIX - Architettura di sistema	65.000	
PP 672	MARKETING CON LOTUS 1 2 3	41.000	F
GY 703	LINGUAGGI DI PROGRAMMAZIONE	69.000	
BE 713	80286 HARDWARE	65.000	
M 679	DIRECT MARKETING	35.000	
M 706	TECNICHE DI MARKETING	43.000	
M 726	DIRECT MAILING	35.000	
	NOVITÀ DICEMBRE '88		
TP 705	ANALISI A-B-C CON dBASE III PLUS	50.000	F
TP 696	CHECK-UP AZIENDALE	75.000	F
PP 698	LOTUS 1-2-3 E dBASE III TRASF. DATI	40.000	F
BE 712	80286 PROGRAMMATORE	55.000	
BE 617	ALIMENTATORI - REGOLATORI SWITCHING	49.000	
BE 739	IL MANUALE DEL TIMER 555	21.000	
GE 646	CORSO DI ELETTRONICA	78.000	
TP 785	dBASE III E PLUS IN UFFICIO	50.000	F

CODICE	TITOLO	PREZZO	
	NOVITÀ GENNAIO '89		
R 720	IL MANUALE DEL GW Basic	45.000	F
PP 719	POSTSCRIPT	60.000	
GT 704	RETI DI COMPUTER	75.000	
GY 702	PROGETTAZIONE E SVILUPPO DEI SISTEMI OPERATIVI	69.000	
BE 711	80386 PROGRAMMATORE	55.000	
PP 752	VIDEOSCRITTURA 4	59.000	
CL 757	AMIGA	59.000	F
R 759	SCHEDA GRAFICA EGA	43.000	2F
R 781	POWER WINDOWS 2.0 E 386	53.000	
GE 794	ARCHITETTURA RISC	32.000	
CZ 805	3½" MS-DOS SOFTWARE	35.000	4F
CZ 806	PC FLOPPY	25.000	4F
CZ 807	PC SOFTWARE	25.000	4F
	GRANDI OPERE		
159B	E.I.	695.000	
162SFR	SOFTWARE	360.000	
161R	D.E.I.	460.000	
160B	ABC	215.000	
LB02E	LABORATORIO DI ELETTRONICA	360.000	
BY02E	BYTES	505.000	
EE02E	ELETTRICITÀ & ENERGIA (disp. Febbraio '89)	295.000	
161RM	ENCICLOPEDIA MONOGRAFICA DI ELETTRONICA E INFORMATICA	190.000	

GRUPPO EDITORIALE JACKSON
DIVISIONE PUBBLICITÀ

F = libro con floppy
C = libro con cassette

Per le vostre ordinazioni per corrispondenza utilizzate l'apposita cedola inserita in questa rivista.

Y0008/D

TRANSACTOR PER AMIGA N.2

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.

Lo sconto del 40% è stato calcolato, in certi casi, arrotondando le cifre in modo da differenziare le tariffe di ciascuna rivista per esigenze di gestione.
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. 60.500
☐ AUTOMAZIONE OGGI	numeri 20	L. 60.000
☐ MECCANICA OGGI	numeri 40 + 6 omaggio	L. 59.000
☐ STRUMENTAZIONE E MISURE OGGI	numeri 11	L. 39.000
☐ INFORMATICA OGGI SETTIMANALE	numeri 40 + 6 omaggio	L. 61.000
☐ INFORMATICA OGGI MESE	numeri 11	L. 33.500
☐ BIT (quindic. da Gennaio)	numeri 20	L. 48.000
☐ PC MAGAZINE	numeri 11	L. 32.500
☐ PC FLOPPY	numeri 11	L. 79.500
☐ COMPUTERGRAFICA E APPLICAZIONI	numeri 11	L. 39.500
☐ TRASMISSIONE DATI E TELECOM.	numeri 11	L. 34.000
☐ COMPUSCUOLA	numeri 10	L. 24.500
☐ WATT (quindic. da Gennaio)	numeri 20	L. 36.500
☐ MEDIA PRODUCTION	numeri 11	L. 46.500
☐ STRUMENTI MUSICALI	numeri 11	L. 32.000
☐ FARE ELETTRONICA	numeri 12	L. 36.000
☐ AMIGA MAGAZINE disk	numeri 11	L. 92.500
☐ AMIGA TRANSACTOR	numeri 6	L. 25.500
☐ COMMODORE PROFESS. 64/128 disk	numeri 11	L. 85.000
☐ COMMODORE PROFESS. 64/128 cass.	numeri 11	L. 59.500
☐ SUPERCOMMODORE 64/128 disk	numeri 11	L. 79.000
☐ SUPERCOMMODORE 64/128 cassetta	numeri 11	L. 49.500
☐ OLIVETTI PRODEST USER	numeri 6	L. 18.000
☐ PC SOFTWARE	numeri 11	L. 66.000
☐ PC GAMES 5¼"	numeri 11	L. 93.000
☐ PC GAMES 3½"	numeri 11	L. 99.500
☐ 3 ½" SOFTWARE	numeri 11	L. 99.000

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 E GRANDI OPERE

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 L. 40.000 + 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 E FASCICOLI GRANDI OPERE

ORDINI QUI SOTTO CIÒ CHE HA SCELTO

ARRETRATI RIVISTE

☐ 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 ¼	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.

ARRETRATI GRANDI OPERE

☐ ABC PERSONAL COMPUTER (24 fascicoli 4 copertine)	fasc. n. ____ L. 3.000 cad.	cop. n. ____ L. 7.000 cad.
☐ ABC PERSONAL COMPUTER QUIZ (10 fascicoli 1 copertina)	fasc. n. ____ L. 3.000 cad.	cop. n. ____ L. 7.000 cad.
☐ A SCUOLA DI SCACCHI C64/C128 (10 lezioni 1 raccoglitore)	lez. n. ____ L. 8.000 cad.	racc. n. ____ L. 7.000 cad.
☐ BYTES (60 fascicoli 7 copertine)	fasc. n. ____ L. 3.000 cad.	cop. n. ____ L. 7.000 cad.
☐ CORSO DI GRAFICA C64/C128 (10 lezioni 1 raccoglitore)	lez. n. ____ L. 8.000 cad.	racc. n. ____ L. 7.000 cad.
☐ D.E.I. (110 fascicoli 12 copertine)	fasc. n. ____ L. 4.500 cad.	cop. n. ____ L. 7.000 cad.
☐ ELETTRICITÀ E ENERGIA (52 fascicoli 4 copertine)	fasc. n. ____ L. 3.000 cad.	cop. n. ____ L. 7.000 cad.
☐ E.I. 1a Edizione (51 fascicoli 8 copertine)	fasc. n. ____ L. 3.000 cad.	cop. n. ____ L. 7.000 cad.
☐ E.I. 2a Edizione (61 fascicoli 7 copertine)	fasc. n. ____ L. 3.000 cad.	cop. n. ____ L. 7.000 cad.
☐ E.I. AGGIORNAMENTI (20 fascicoli 2 copertine)	fasc. n. ____ L. 3.000 cad.	cop. n. ____ L. 7.000 cad.
☐ LABORATORIO DI ELETTRONICA (52 fascicoli 5 copertine)	fasc. n. ____ L. 3.000 cad.	cop. n. ____ L. 7.000 cad.
☐ LA GRANDE GUIDA DEL PROGRAMMATORE ☐ 128 ☐ 64 (8 fascicoli)	fasc. n. ____ L. 8.000 cad.	
☐ PC MASTER MS DOS (5 1/4) (9 lezioni 1 copertina)	lez. n. ____ L. 14.000 cad.	cop. n. ____ L. 8.000 cad.
☐ PC MASTER MS DOS (3½) (9 lezioni 1 copertina)	fasc. n. ____ L. 17.000 cad.	cop. n. ____ L. 8.000 cad.
☐ D. BASE III (5 1/4") (8 fascicoli 1 copertina)	fasc. n. ____ L. 14.000 cad.	cop. n. ____ L. 8.000 cad.
☐ D. BASE III (3½) (8 fascicoli 1 copertina)	fasc. n. ____ L. 17.000 cad.	cop. n. ____ L. 8.000 cad.
☐ QUADERNI JACKSON (30 fascicoli)	fasc. n. ____ L. 7.000 cad.	
☐ SOFTWARE (52 fascicoli 5 copertine)	fasc. n. ____ L. 2.500 cad.	cop. n. ____ L. 8.000 cad.
☐ VIDEO BASIC (20 lezioni 5 raccoglitori) ☐ C64 ☐ C128 ☐ VIC20 ☐ MSX	lez. n. ____ L. 8.000 cad.	racc. n. ____ L. 8.000 cad.
☐ 7 NOTE BIT (15 lezioni 3 raccoglitori)	lez. n. ____ L. 10.000 cad.	racc. n. ____ L. 10.000 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

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.

ABBONAMENTO GRATUITO

A 6 NUMERI, A SCELTA TRA LE SEGUENTI RIVISTE SETTIMANALI

☐ EO News Sett. ☐ INFORMATICA Oggi Sett. ☐ MECCANICA Oggi (da febbraio '89)

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. ☐ MECCANICA Oggi (da febbraio '89)

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. ☐ MECCANICA Oggi (da febbraio '89)

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

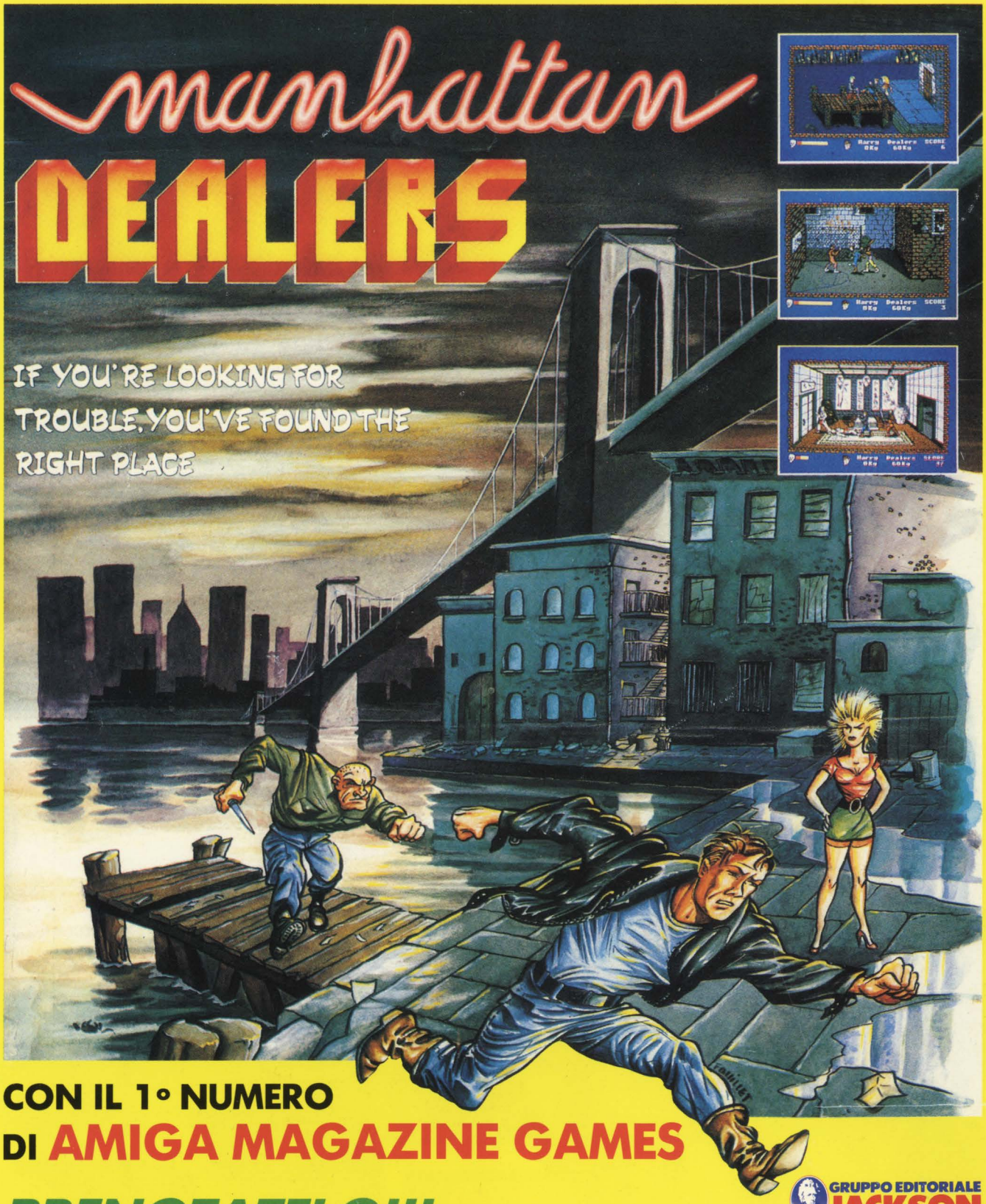
LE RIVISTE JACKSON PER IL TUO COMMODORE

IN EDICOLA O IN ABBONAMENTO: AMIGA MAGAZINE CON FLOPPY • AMIGA TRANSACTOR • COMMODORE PROFESSIONAL 64/128 CON FLOPPY E CASSETTA • SUPER COMMODORE 64/128 CON FLOPPY E CASSETTA

002P-10-88

NON PERDETE D'OCCHIO LA VOSTRA EDICOLA

È IN ARRIVO...



CON IL 1° NUMERO
DI AMIGA MAGAZINE GAMES

PRENOTATELO!!!

GRUPPO EDITORIALE JACKSON
AREA CONSUMER