L. 7.000 - S.FR. 10,50

Amiga Transactor

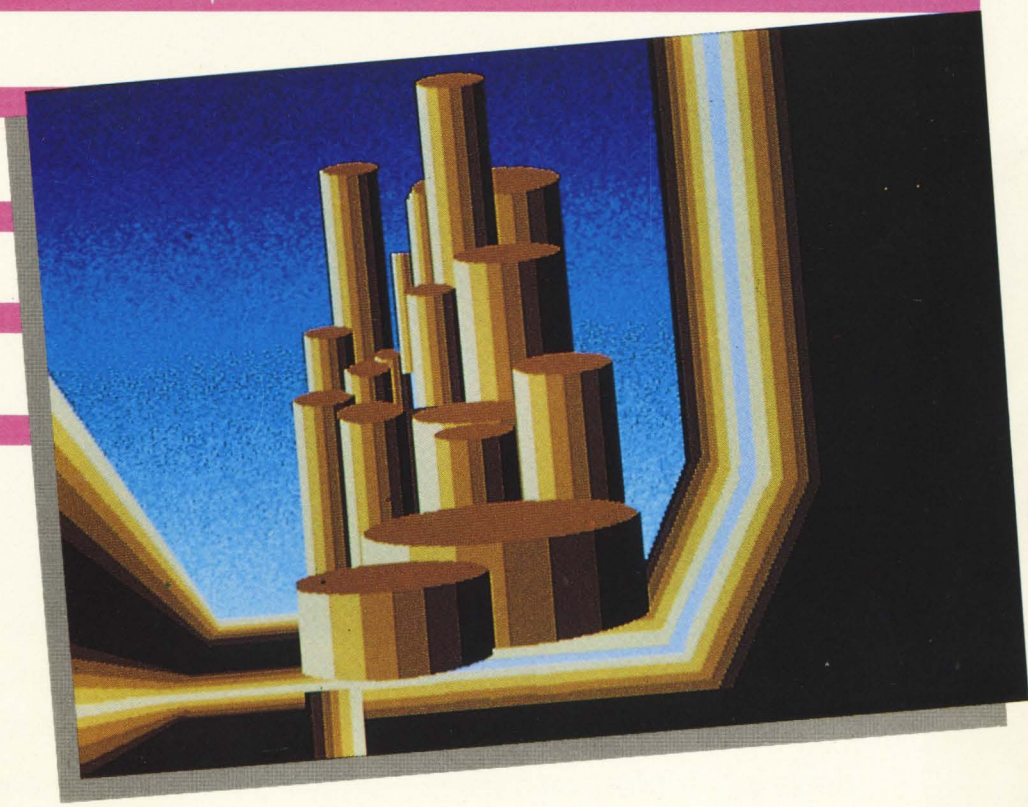
PER

EDIZIONE ITALIANA

NUMERO 6 1989

LA RIVISTA PER I PROGRAMMATORI DI AMIGA

- Viewport
- Programmare in Arexx (parte 2)
- I messaggi di Amiga
- Lo standard IFF
- Algoritmo di riempimento veloce
- Linguaggio Assembly (parte 2)
- Gadget multipli
- L'arte della programmazione in linguaggio Assembly
- Disegnare con la libreria grafica
- I dischi di Fred Fish



GRUPPO EDITORIALE JACKSON

TAXE PERCUE (TASSA RISCOSSA) MILANO FERR. CORR. SPEDIZIONE IN ABB. POST. GRUPPO IV/70

Transactor arrivederci?

Il titolo sembrerebbe preannunciare qualcosa di poco lieto. Dipende dai punti di vista.
Il Gruppo Editoriale Jackson è stato il primo e l'unico a proporre una serie di pubblicazioni che si stagliassero immediatamente dalla piatta media italiana. Non vorremmo peccare d'orgoglio ma, onestamente, quale altra rivista italiana può vantare l'autorevolezza e la competenza di Transactor. Quale editore è riuscito a pubblicare una collana di ben 9 libri esclusivamente dedicati ad Amiga dei quali 6 addirittura nell'ultimo anno? Chi ha dato la possibilità a qualunque italiano di poter ricevere a casa propria il meglio della produzione mondiale di software di pubblico dominio (leggi Fish Disk)?
E il mercato? Voi, come avete risposto?
Il bilancio del primo anno di vita di Transactor è decisamente lusinghiero, avendo raggiunto volumi di vendita superiori a quelli preventivati, soprattutto se si pensa ai non piccoli problemi che abbiamo dovuto affrontare nella primavera scorsa quando siamo stati sull'orlo della chiusura per l'interruzione dei rapporti con gli autori esteri. Se nel nostro piccolo possiamo cantare vittoria, così non può essere se si considera la realtà nazionale. All'inizio di questa avventura pensavamo che, grazie anche al nostro contributo, l'utente medio italiano (il Moro di Canale, tanto per intenderci) sarebbe uscito dal torpore che ne aveva caratterizzato da sempre l'esistenza. In cuor nostro pensavamo che i politici di casa, sempre molto attenti al campanile, si sarebbero affrettati a introdurre, essendo rimasti gli ultimi in Europa, una legge che proibisse la copia illegale del software. In sostanza la speranza era che si riuscisse a imboccare una strada che ci potesse portare fuori dal terzo mondo.
I Ladri continuano invece a scorazzare indisturbati, le edicole sono piene di riviste indecenti (ne esistono anche, poche, di decenti e ottime), gli Amiga fanno bella mostra di sé insieme ai tostapane e ai ferri da stiro e il, poco, software originale reperibile è in massima parte composto da giochi. Il Moro di Canale non si è potuto muovere, anche se avrebbe voluto, e anzi ha incontrato più difficoltà di prima.
In questo scenario e dopo lunghe e animate discussioni si è giunti alla conclusione che una rivista sola sarebbe stata meglio di due. Di conseguenza il Gruppo Editoriale Jackson raggruppa sotto un'unica testata Amiga Magazine e Transactor. La nuova rivista, che manterrà il nome della più vecchia (Amiga Magazine) conterrà al suo interno un inserto contenente gli articoli di Transactor con gli usuali servizi. La rivista avrà cadenza mensile e sarà corredata di dischetto, esaudendo finalmente le richieste di quanti desideravano vedere Transactor completa di listati su disco.
Per quanto mi riguarda vi saluto con molto affetto, passando ad altri la redazione e la gestione di tale super-rivista, certo comunque che questo non si tratti di un addio. Appena la situazione si sarà un po' schiarita, ovvero si comincerà a uscire dal terzo mondo, Transactor potrà tornare a contare sulle sole proprie forze. Arrivederci, quindi.

Marco Ottolini

per AMIGA Transactor
La rivista dei programmatori di Amiga

Sommario

DIRETTORE RESPONSABILE
Paolo Reina

DIRETTORE TECNICO
Marco Ottolini

TRADUZIONI E REDAZIONE
Leonardo Fei, Stefano Simonelli,
Paolo Toccaceli, Giovanna Traversa

GRAFICA
Roberto Pessina

IMPAGINAZIONE ELETTRONICA
Roberto Gibertini

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

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

GRUPPO EDITORIALE JACKSON
DIVISIONE PERIODICI

AREA CONSUMER
PUBLISHER
Filippo Canavese

DIREZIONE
Via Pola, 9 - 20124 Milano
Tel. 02/69401
Telefax 02/6948238

REDAZIONE, PUBLICITA', AMMINISTRAZIONE, ABBONAMENTI
Via Rossellini, 12 - 20124 Milano
Tel. 02/69401
Telex: 333436 GEJIT
Telefax: 02/6948489

CONCESSIONARIO ESCLUSIVO
SODIP - Via Zuretti, 25 - 20125 Milano

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

SEDE LEGALE
Via Pietro Mascagni, 14 - 20122 Milano

PREZZO DELLA RIVISTA: L. 7.000
NUMERI ARRETRATI: L. 14.000

Spedizione in Abb. Post. Gruppo IV/70

PUBLISHER
David H. Beatty

EDITOR
Mike Todd

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

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

Soundware: è tutto oro quel che luccica?

Uno sguardo ravvicinato con nuovi operatori

Prima di tutto una doverosa spiegazione sulla mancanza dello spazio "Lettere & Bit" in questa pagina: data la situazione di prossima chiusura della testata, ci è parso più utile illustrare una novità del mercato italiano piuttosto che trattare temi futuri, come proposti dalla maggior parte delle lettere giunte in redazione. Un passo, per cercare di uscire dal "terzo mondo".

Soundware, al contrario di quello che il nome potrebbe portare a pensare, non è l'ultimo software musicale che si può trovare sotto l'albero di Natale, ma il nome di una nuova società che si distingue, per alcuni versi, nel panorama di quelle che si occupano di accessori, hardware e/o software per computer. Ha sede a Varese e propone hardware e software, per il momento, riservato ad applicazioni musicali e video. Il prodotto più importante, per Amiga, distribuito è sicuramente quel Music X, della Microillusions, che da molti parti viene indicato come il migliore software in circolazione.

Al di là dei prodotti commercializzati, la Soundware, cerca di imporre una visione un po' più professionale del vendere software per Amiga. Gli stessi ideatori della società cercano di porre l'accento su alcune iniziative, uniche nel loro genere in Italia, come veri e propri veicoli per invogliare all'acquisto piuttosto che per giustificare l'alto prezzo di vendita. Per esempio il manuale del Music X, oltre che essere interamente in italiano è stato arricchito di vari contributi che ne fanno una vera e propria guida alla produzione musicale per mezzo di computer e MIDI.

Music X

Merita sicuramente un posto di rilievo questo pacchetto che promette meraviglie. La prima cosa da mettere in chiaro è che non è un programma adatto a chiunque, è indicato infatti per una utenza professionale o semi-professionale, indicando con questa dizione anche chi, pur essendo un amatore, ha già maturato alcune esperienze significative. Brutte notizie quindi per chi si aspettava un *Instant Music* o un *Deluxe Music* potenziato. Il secondo appunto riguarda il fatto che, pur funzionando anche con le voci interne di Amiga, è indicato per l'uso di Amiga in unione a una o più apparecchiature MIDI.

Music X è un programma composto da diversi moduli, in grado di interagire tra di loro, che svolgono una serie di compiti propri di una vera e propria workstation musicale. Il sequencer agisce come un registratore mutitraccia dotato di 20 tracce e 250 sequenza, ognuna editabile e modificabile separatamente. Il librarian permette di gestire i suoni interni di un sintetizzatore MIDI, potendoli caricare da e verso Amiga. Il Music X è dotato anche di un editor generico per gestire i parametri dei diversi sinth collegati via MIDI. Sono predisposti già alcuni moduli specifici per i sintetizzatori più diffusi, ma è sempre possibile, servendosi di un ulteriore modulo, costruire un editor perticolare per il proprio strumento.

Lo spazio è tiranno e non ci permette di andare oltre; basterà comunque dire che il programma è realmente unico e che il costo è di £ 399.000. Troppo? Considerando il pubblico a cui è rivolto, che il prodotto è assistito, che le istruzioni sono in italiano, che è dotato di una interfaccia MIDI e di una guida alle applicazioni musicali non si può certo dire di si.

AmiGen e Midia MusicBox

Un altro prodotto Soundware interessante è sicuramente AmiGen, un genlock per Amiga prodotto dalla Mimetics, ditta famosa fin dagli albori di Amiga per un digitalizzatore audio e per il software ProMIDI. L'importante da sapere su questo genlock è il fatto di essere di ridotte dimensioni, il che non guasta mai, di elevate prestazioni e di basso prezzo. È disponibile sia nella versione NSTC che PAL ed è dotato anche di un connettore RGB thru

Concludiamo la panoramica sui prodotti Soundware, senza dimenticare che distribuisce anche i prodotti DR.T'S (tra cui il famoso KCS), dando uno sguardo al Midia MusicBox. Si tratta di un expander MIDI, cioè di un sintetizzatore a tutti gli effetti ma senza tastiera musicale, di buone caratteristiche espressamente pensato per essere usato in unione a un computer, cui si collega direttamente. È dotato di 1029 tipi di suoni, dispone di uscite mono e stereo e ha un mixer digitale dentro di sé. In effetti potrebbe rappresentare il degno contraltare di Music X per chi voglia esplorare nuovi terreni musicali.

Soundware, Via Mazzini 12, 21020 Casciago (VA)
Tel. 0332/222052 - Fax 0332/228288

Scopri i segreti di AMIGA

Novità

DOS Versione 1.3

R. Kerkloh, M. Tornsdorf, B. Zoller
Il testo analizza esaurientemente tutti i comandi della versione 1.3 dell'AmigaDOS.
Cod. CC815 pp.336 L. 59.000
Con floppy disk 3 1/2"

per imparare il linguaggio C con Amiga

Edgar Huckert, Frank Kremser
Per sfruttare le enormi potenzialità grafiche del proprio computer, attraverso un linguaggio di programmazione adatto a questo tipo di applicazioni.
Cod. CL758 pp.208 L. 52.000
Con floppy disk 3 1/2"

Henning Horst-Rainer
Introduce alla programmazione in AmigaBASIC presentando 100 programmi ed esempi di utilizzo degli oltre 200 comandi del BASIC.
Cod. CL768 pp.320 L. 57.000
Con floppy disk 3 1/2"

per sfruttare tutte le potenzialità grafiche di Amiga

Axel Plenge
Per apprendere la progettazione, la programmazione e la rappresentazione su Amiga di grafici e immagini tridimensionali.
Cod. CZ756 pp.368 L.59.000
Con floppy disk 3 1/2"

Robert A. Peck
Contiene una disamina delle tecniche avanzate di programmazione e di ottimizzazione nell'utilizzo dei linguaggi più evoluti.
Cod. CC795 pp.430 L. 62.000
Con floppy disk 3 1/2"

Peter Wollschlaeger
Nessuna limitazione alle potenzialità di Amiga quando il linguaggio di programmazione è l'Assembler.
Cod. CL757 pp.324 L. 59.000
Con floppy disk 3 1/2"

SUL MEDESIMO ARGOMENTO

D. Lawrence, M. England
AMIGA HANDBOOK
Cod. CC320 pp.200 L. 35.000

R. Bonelli, M. Lunelli
AMIGA 500
Guida per l'utente
Cod. CC627 pp.376 L. 55.000

A. Bigiarini, P.L. Cecioni, M. Ottolini
IL MANUALE DI AMIGA
Cod. CZ532 pp.244 L. 39.000

AM1.1/1

GRUPPO EDITORIALE JACKSON

I libri del Gruppo Editoriale Jackson sono in vendita presso le migliori librerie e computershop. Se ti è più comodo acquistarli per corrispondenza utilizza questo coupon.

Da spedire in busta chiusa a: **GRUPPO EDITORIALE JACKSON, Via Rosellini 12 - 20124 Milano**

Si, inviatemi i volumi sottoelencati

38

INDICARE CHIARAMENTE CODICI E QUANTITA' DEI VOLUMI RICHIESTI

Codice	Q.ta	Codice	Q.ta	Codice	Q.ta	Codice	Q.ta	Codice	Q.ta

Ordine minimo L. 60.000 + L.4.500 per contributo fisso spese di spedizione

❑ Sono titolare Gold Card n°: G ❑ e ho diritto allo sconto del 10% (fino al 31/12/89)

❑ Non sono titolare

Modalità di pagamento:

❑ Contro Assegno postale al ricevimento dei volumi

❑ Assegno allegato n° _____ Banca _____

❑ Ho effettuato il pagamento a mezzo: ❑ Vaglia postale ❑ Versamento sul c/c post. n° 11666203 a Voi intestato e allego fotocopia della ricevuta

❑ Addebitatemi l'importo di L. _____ sulla carta di credito: ❑ Visa ❑ American Express ❑ Diners Club ❑ Carta Si

Conto n° _____ scadenza _____

❑ Richiedo fattura (Partita IVA n° _____)

Cognome e Nome _____

Via _____ n° _____

Cap _____ Città _____ Prov. _____

Tel. _____ Data _____ Firma _____

Programmare in ARexx

Parte 2 - decisioni e loop

di John Carpenter

Il set di istruzioni di Rexx risulta piccolo quando lo si paragona con alcuni altri linguaggi, ma quello che c'è viene affiancato da un grande numero di funzioni (un concetto familiare ai programmatori C). Coloro che cominciano a usare Rexx, spesso si confondono per la differenza che Rexx fa per istruzioni, variabili e funzioni.

Le *funzioni* sono sempre seguite da un suffisso fra parentesi e normalmente restituiscono un valore, come in *l* = Length(string). In questo caso *l* è una variabile alla quale viene assegnata la lunghezza della stringa contenuta nella variabile *string*.

Un'*istruzione* è una parola chiave che *non* è seguita da un due punti o dal segno di uguale. Se la parola chiave *è* seguita da un due punti allora diventa un'*etichetta*. Se invece viene seguita dal segno di uguale allora diventa una *variabile*.

Da tutto questo, segue che non esistono parole chiave riservate ma che l'utilizzo di parole chiave che normalmente rappresentano un'istruzione come variabili o etichette causerà verosimilmente una grande confusione nel lettore.

Prendiamo il seguente programma come esempio (notate che il numero di linea negli esempi svolge puramente il compito di facilitare il riferimento a una linea specifica e che *non* fa parte del programma stesso):

```
1: /* I programmi Rexx devono iniziare con /* */  */
2: Say = "Say"
3: Say Say
4: Exit 0
```

Il risultato di questo programma sarà la visualizzazione della parola "Say". Nella seconda linea, *Say* è una variabile, mentre nella linea 3, la prima *Say* rappresenta un'istruzione e la seconda è di nuovo la variabile. La prima istruzione che abbiamo incontrato era SAY. La sintassi è:

```
SAY [espressione]
```

Le parentesi quadre indicano che i contenuti sono opzionali. SAY, usata da sola, manderà il carattere di newline alla console. L'espressione può contenere un numero qualsiasi di stringhe, variabili e funzioni. Se non riusciamo a fare stare l'espressione su una linea, possiamo usare una virgola per indicarne la continuazione sulla linea successiva.

```
1: /* */
2: Say "Ciao a tutti." "Che bel",
3: "linguaggio è questo Rexx!"
4: Exit 0
```

Il programma precedente dà come risultato:

```
Ciao a tutti. Che bel linguaggio è questo Rexx!
```

Quello che potrebbe sorprendervi nell'output di questo programma è la presenza di uno spazio fra *tutti.* e *Che.* Dato questo fatto, come si fa a unire insieme stringhe o variabili senza spazi?

```
1: /* */
2: Say "Non""funziona perché include l'apice doppio"
3: RC=0  /* prepara una variabile */
4: Say 'Questo invece sì, codice di return='RC
5: Say "Anche"||"questo."
6: Exit 0
```

La linea 2 provocherà l'apparizione di un doppio apice nel bel mezzo dell'output. Se la stringa fosse stata delimitata da un singolo apice, la parola l'apice avrebbe dovuto essere codificata come l''apice

La quarta linea non risente di questo problema, quindi appendendo la variabile RC alla fine della stringa produciamo l'effetto desiderato. Avremmo anche potuto usare l'operatore di concatenazione di stringa (||) come nella linea 5.

Prendere delle decisioni

La sequenza IF..THEN..ELSE è simile a quella del BASIC e la sua sintassi è:

```
IF espressione THEN [istruzione]
```

Se l'espressione è vera, allora l'istruzione verrà eseguita. Il comando THEN non deve necessariamente essere sulla stessa linea dell'IF. Se desideriamo eseguire un'istruzione vuota, utilizziamo il codice NOP (no-operation).

```
1: /* */
2: a=3
3: If a=1 then Nop  /* esegue un NOP */
4: Else Say "a è diverso da 1"
5: If a=3 then Say "a è uguale a 3"
6: Exit 0
```

La linea 3 esegue un'istruzione nulla, mentre la linea 4 contiene l'istruzione reale. Questa tecnica evita di usare la logica inversa che spesso porta a degli errori. La linea 5 mostra che il comando ELSE non è essenziale.

Variazioni in DO

Per eseguire un gruppo di istruzioni, anziché le singole istruzioni come abbiamo visto finora, dobbiamo usare i comandi DO..END. DO indica l'inizio di un blocco, mentre END ne costituisce il delimitatore finale.

```
1: /* */
2: a=3
3: If a=1 then Nop  /* esegue un NOP */
4: Else
5:   Do
6:     Say "a non è uguale a 1"
7:     Say "ma è uguale a" a
8:   End
9: If a>0 then
10:  Do
11:    Say "a>0"
12:    If a=1 then Break
13:    Say "a>1"
14:  End
15: Exit 0
```

In questo esempio, entrambe le linee 6 e 7 verranno eseguite come un solo blocco.

Il blocco DO..END successivo mostra l'uso dell'istruzione BREAK. Questa viene usata per uscire prematuramente da un blocco.

Così com'è organizzato il programma, entrambe le istruzioni SAY verranno eseguite; ma se a è uguale a 1, allora verrà eseguito il primo SAY e poi verrà eseguito il BREAK. Questo provocherà il salto del successivo SAY e riprenderà l'esecuzione del programma da dopo l'istruzione END, trovando subito EXIT.

La coppia DO..END non viene usata solo per delimitare un blocco di istruzioni ma può venire utilizzata anche per controllare un loop. Il BASIC naturalmente dispone anche dei costrutti FOR..NEXT e FOR..NEXT..STEP. Rexx ha un costrutto equivalente nella forma di una variazione del loop DO. È formato dai comandi DO..TO e DO..TO..BY e la sua sintassi è:

```
DO var=espressione TO espressione [BY espressione]
```

Nel progamma seguente il primo blocco DO..END stampa i numeri da 1 a 5, mentre il secondo stampa quelli da 5 a 1.

```
1: /* */
2: Do i=1 to 5
3:   Say i
4: End
5: Do i=5 To 1 By -1
6:   Say i
7: End
8: /* DO può essere sia delimitatore di blocco */
9: /* per IF/THEN sia controllore di loop */
10: a=3
11: If a>0 then
12:   Do i=1 To a
13:     Say i
14:   End
15: /* questo è equivalente a */
16: If a>0 then
17:   Do
18:     Do i=1 to a
19:       Say i
20:     End
21:   End
22: Exit 0
```

Un altro costrutto di controllo per costruire cicli è formato da DO WHILE..END, la cui sintassi è:

```
DO WHILE espressione
```

L'espressione deve restituire un valore booleano. Se il risultato è vero, allora l'iterazione procede, se falso termina. L'espressione viene valutata all'inizio del loop.

Al contrario, DO UNTIL..END, un altro costrutto per il controllo di cicli, valuta l'espressione condizionale alla fine del loop che quindi verrà eseguito sempre almeno una volta. La sua sintassi è:

```
DO UNTIL espressione
```

Il seguente programma dimostra l'uso delle istruzioni WHILE e UNTIL:

```
1: /* */
2: a=0
3: Do While a<3
4:   Say "While"
5:   a=a+1
6: End
7: Do Until a=3
8:   Say "Until"
9: End
10: Exit 0
```

Il risultato è che *While* viene stampato tre volte e *Until* una.

Circoli chiusi

L'utima variante di DO..END è DO FOREVER. Questo costrutto ricorda quello che normalmente è l'incubo di un programmatore, ma in realtà si dimostra molto utile. Il controllo del loop si ottiene mediante l'istruzione LEAVE. Quando quest'istruzione viene eseguita, l'esecuzione passa all'istruzione che segue il comando END. La sua sintassi è:

```
DO FOREVER
LEAVE [variabile]
```

Se l'istruzione LEAVE viene eseguita senza il nome della variabile, allora provocherà l'uscita dal loop DO più interno. Se il nome della variabile viene specificato, si può uscire da loop annidati l'uno dentro l'altro. Quest'istruzione può essere utilizzata con qualsiasi loop DO..END.

```
1: /* */
2: Do i=1 To 3
3:   Do j=1 To 5
4:     Say i "volte" j "uguale" i*j
5:     If i=3 & j=3 then Leave i
6:   End
7: End
8: Exit 0
```

Venogno prodotti, come risultati, la tabella da 1*1 a 1*5 e quella da 2*1 a 2*5. La tabella dei 3 termina a 3*3. In questo punto l'istruzione LEAVE causa l'esecuzione dell'istruzione che segue l'END del ciclo più esterno, cioè il comando Exit.

L'altra istruzione di controllo per loop è la ITERATE.

```
1: /* */
2: Parse Upper Arg FILE  /* caratteristiche del
                            file */
3: If Open('file',FILE,'R') then nop /* apri il
                            file */
4: else do
5:   Say 'Non posso aprire il file' FILE6:   Exit 1
7:   end
8: Do forever  /* loop di lettura */
9:   Rec = Readln('file')  /* leggi una linea */
10:  If Eof('file') then leave
11:     /* se questa è la fine del file, esci */
12:  If Upper(Substr(Rec,1,1)) ~= 'A' then Iterate
13:     /* se primo carattere != da a o A
                            ricomincia */
14:  Say Rec  /* output */
15:  end
16: a = Close('file')
17: Exit 0
```

Questo programma cerca una linea che cominci con il carattere A o a. Se non lo trova subito, allora salta alla fine del loop DO e legge un'altra linea. La linea che contiene l'istruzione ITERATE è interessante in quanto contiene una funzione nidificata. Upper(Substr(Rec,1,1)) inizialmente ottiene il primo carattere dalla variabile Rec e quindi converte il risultato in forma maiuscola.

L'istruzione ARG

Nell'esempio precedente, ARG viene usata per leggere gli argomenti passati al programma utilizzando la linea:

```
Parse Upper Arg FILE
```

Questa è composta, in realtà, da tre istruzioni: precisamente PARSE, UPPER e ARG. Se controllate nel manuale, vi accorgerete che avrei potuto limitarmi a scrivere ARG. In questa istruzione particolare, tutti gli argomenti verrebbero passati dentro la variabile FILE.

Interpretando le espressioni

L'istruzione INTERPRET è abbastanza recente. La sintassi è:

```
INTERPRET espressione
```

L'espressione viene prima valutata e quindi eseguita. Se l'espressione contiene una o più variabili allora queste vengono valutate per l'esecuzione.

```
1: /* */
2: a = 'Say "Il gatto dormiva sullo zerbino"; Say
                                    Date()'
3: Interpret a 'today'
4: Drop a
5: If Open('file','TEST','R') then nop /* apri il
file */
6: else do
7:   Say 'Non posso aprire il file' FILE8:   Exit 1
9:   end
10:  Do forever      /* loop di lettura */
11:    Rec = Readln('file')  /* leggi una linea */
12:    If Eof('file') then leave
13:       /* se fine del file, esci dal loop */
14:    Interpret Rec  /* esegui espressione in Rec*/
15:    end
16: a = Close('file')
17: Exit 0
```

Se creiamo un file ASCII chiamato TEST, contenente le seguenti linee:

```
Say 'Ciao a tutti'
a = 'Say "interpretazione annidata"'
Interpret a  /* funziona anche questo */
```

e poi eseguiamo il programma, otterremo:

```
Il gatto dormiva sullo zerbino
10 Mar 1989 TODAY
Ciao a tutti
interpretazione annidata
```

La serie proseguirà su Amiga Magazine, con un'analisi del PARSING e della chiamata di altri programmi e funzioni.

ViewPort

In lode al Basic... ma non all'AmigaBasic

di Larry Philips

Se chiedete a un utente Amiga quali sono le sue conoscenze di programmazione la risposta sarà una delle più disparate. Scopriremo background di tutti i generi, da coloro che hanno comperato per la prima volta un computer, a quelli che hanno vasta esperienza su mainframe, dai programmatori di sistema a utenti finali, dai principianti agli esperti. Molte di queste categorie si trovano bene nell'ambiente di Amiga, in modo particolare per quanto riguarda i programmi e i linguaggi di programmazione.

A proposito della programmazione di Amiga, l'utente ha molti tool a disposizione, inclusi assembler e compilatori C, Fortran, Lisp, Pascal, Modula-2, APL e così via. I programmi applicativi sono abbondanti e soddisfano gli interessi più svariati. Esiste, tuttavia, un'area che potrebbe essere notevolmente migliorata.

Non tutti amano la programmazione e ci sono anche coloro che rabbrividiscono al solo pensiero di scrivere un programma di qualsiasi tipo, anche un semplice file script o batch. L'utente Amiga che si trova in questa posizione, deve accontentarsi dei programmi già esistenti, aspettare lo sviluppo di nuovi applicativi, o preoccuparsi attivamente di convincere un programmatore che il desiderato applicativo interesserebbe anche altri.

Dall'altra parte ci sono i veri programmatori, gli sviluppatori di software applicativo, di utilità, di giochi e di altro ancora. Costoro sono i cosiddetti 'fanatici della programmazione' che provano tutti i linguaggi a loro disposizione. In realtà, alcune di queste persone posseggono un computer per nessun'altra ragione se non quella di scrivere programmi. Sono le persone responsabili della scrittura dei molti programmi che sono disponibili per Amiga. In genere non scrivono programmi senza una ragione ben precisa. Normalmente sono motivati da una personale esigenza di un particolare tipo di programma o dalla prospettiva di ottenere dei vantaggi economici dalla vendita del programma stesso.

Fra questi due estremi, si trovano molti utenti Amiga che hanno la capacità e il desiderio di creare programmi. Questa è la fascia per la quale potrebbero essere migliorate le cose. Esiste, infatti, un gran numero di utenti Amiga che scriverebbero programmi piccoli e semplici, se ci fossero i necessari strumenti a disposizione. Potrebbero scrivere quei piccoli programmi di utilità che nessun altro sembra avere voglia di scrivere. Questi potrebbero essere programmi molto mirati, che sono utili solo a una persona. Partendo da questi, potrebbero diventare più ambiziosi, trasformandosi, col passare del tempo, in programmatori di applicativi di pubblica utilità. Non tutti sono disposti a comperare un compilatore o a imparare un nuovo linguaggio per arrivare a questi obiettivi.

Immaginate la persona che possedeva un C64, Atari 800, IBM PC o qualsiasi altro piccolo computer, che avesse scritto un certo numero di programmi in BASIC e che volesse fare lo stesso sull'Amiga. Apre la scatola di Amiga, collega tutto e la prima volta che vuole mettersi a scrivere quella piccola utility, sbatte violentemente contro il muro dell'AmigaBasic. Non so come sia l'ambiente Basic sul PC IBM o sull'Atari (o della maggior parte dei micro a 8 bit), ma posso dirvi che quello del C64 è lusso allo stato puro, quando lo si paragona a quello dell'AmigaBasic.

Ebbene, quegli utenti che vogliono scrivere un programma ogni tanto perché ne hanno bisogno possono adattarsi alle deficienze dell'interfaccia utente dell'AmigaBasic, ma può rassegnarsi colui che avrebbe scritto volentieri un programma per puro divertimento?

Di sicuro non è molto divertente osservare le finestre mentre si ridisegnano o rendersi conto, appena dato il via a un programma, che c'è un errore di sintassi e che dobbiamo aspettare che il requester appaia, per compiere quella che per altro è l'unica mossa possibile, prendere il mouse, cancellare il requester e attendere che la finestra dell'errore se ne vada finalmente.

AmigaBasic in sé stesso, come ho già avuto modo di dire in queste colonne, è un'implementazione decente di BASIC, ma solo se si possono ignorare i bug e la penosa interfaccia utente. L'unica ragione per cui ripeto questa affermazione è perché ho visto un Basic che è facile da usare, amichevole e compatibile con l'AmigaBasic (senza averne ereditato i bug, naturalmente).

Questo Basic è stato scritto dalla HiSoft, una compagnia che produce software anche per i computer Atari. Il pacchetto è chiamato 'HiSoft Basic Professional' e tiene fede al proprio nome in molti modi. Il Basic della HiSoft fornisce un ambiente

piacevole da usare, con un editor integrato e un localizzatore di errori che permette di editare, compilare, corregere gli errori e provare il programma senza uscire dall'editor stesso.

È possibile anche generare codice che richiede la presenza di una libreria per funzionare, oppure codice che può essere esguito da solo. Quest'ultima versione di un programma risulta di dimensioni ragionevolmente contenute, in quanto estrae dalla libreria solo le funzioni di run-time necessarie per l'esecuzione di quel determinato programma. I programmi che dipendono dalla libreria sono molto piccoli e le funzioni di run-time sono disponibili in una libreria di 46K che può essere condivisa anche fra più programmi.

Non siamo comunque obbligati a usare l'editor e l'ambiente di sviluppo della HiSoft, quindi possiamo usare il nostro editor preferito e il CLI per scrivere i programmi. Il pacchetto è fornito di un voluminoso manuale che risulta accurato e leggibile. Tutto sommato è un pacchetto ben pensato e implementato.

È difficile parlare della HiSoft senza usare toni entusiastici. Sebbene io non sia esattamente un fanatico del Basic, posso apprezzare un lavoro ben fatto in qualsiasi implementazione di un dato linguaggio. È anche difficile parlarne senza farne una piccola recensione, ma ho voluto farvi sapere le differenze fra l'AmigaBasic e il pacchetto della HiSoft prima di procedere oltre.

Date le enormi differenze in termini di facilità d'uso fra questi due pacchetti, mi fa soffrire il fatto di vedere che l'AmigaBasic viene ancora fornito con la macchina. Mi fa soffrire il fatto che la Microsoft tragga profitto da un qualcosa lasciato a metà. Mi fa soffrire il fatto di pensare a tutti quegli utenti Amiga che, se non fosse stato per l'AmigaBasic, avrebbero potuto scrivere un gran numero di programmi, magari a beneficio della comunità di Amiga in generale e comunque sicuramente a proprio beneficio. Forse è significativo il fatto che ci sia ben poco software di pubblico dominio scritto in AmigaBasic e che questo, come linguaggio, venga evitato da molti programmatori serii.

Circa un anno fa, ho chiesto alla Microsoft, attraverso il loro servizio informazioni su Compuserve, se avevano intenzione di aggiornare il loro prodotto per Amiga. La risposta che ricevetti era essenzialmente che avrei dovuto rivolgere questa domanda alla Commodore. Il chiaro messaggio, anche se non detto esplicitamente, era che a loro semplicemente non interessava.

Adesso, onestamente, può essere anche stato che abbiano rispettato correttamente il loro contratto con Commodore, che il prodotto sia stato accettato e che, per scrivere una versione più aggiornata, la Microsoft richieda un ulteriore pagamento.

Non riesco a immaginare una sola altra ragione perché una compagnia lasci un programma come AmigaBasic senza migliorarlo, con tutto quello che implica per l'immagine della compagnia stessa. Magari a loro non interessa che tutti gli utenti Amiga sappiano quanto scadenti possano essere i prodotti Microsoft, oppure magari pensano che comunque non ci saranno mai abbastanza utenti Amiga perché questo possa fare una qualche differenza.

Pensandoci bene, non sono nemmeno sicuro di desiderare che la Commodore riapra la faccenda, vista l'alternativa offerta da pacchetti come l'HiSoft Professional e viste le passate produzioni della Microsoft su altre macchine. Penso, comunque, che la Commodore dovrebbe prendere in considerazione la possibilità di chiedere una licenza per includere un pacchetto come lo HiSoft Professional in ogni macchina venduta. Considerate i vantaggi che si avrebbero fornendo un Basic veramente buono.

Come accade con il Workbench e il CLI, i linguaggi inclusi insieme all'Amiga sono sotto gli occhi degli utenti effettivi e potenziali di questa macchina. Giusto o sbagliato che sia, molti se ne servono come parametro per giudicare l'utilizzabilità e potenza offerta dal computer.

Dovremmo forse fornire delle scuse quando mostriamo una certa parte di Amiga a qualcuno? Non penso proprio, perché si tratta solo di fare qualcosa per gli aspetti che risultano imbarazzanti da mostrare. Sicuramente l'AmigaBasic rientra perfettamente nella descrizione di quello che è considerato imbarazzante.

Fornire un'implementazione decente del Basic aprirebbe la macchina all'uso di coloro che programmano casualmente. Coloro che approdano ad Amiga venendo dal mondo degli home computer a 8 bit, si sentirebbero subito a proprio agio con un simile linguaggio di programmazione e ne farebbero, molto verosimilmente, un buon uso, magari passando successivamente a linguaggi di programmazione diversi.

La domanda cha adesso viene in mente è "Ci sono altri linguaggi candidati a essere inclusi insieme alla macchina?". La risposta a questa domanda è sicuramente *sì*. I sostenitori di ARexx vi diranno che questo lingaggio li serve al pari e meglio di quello che potrebbe fare mai una qualsiasi implementazione del Basic.

Un'altra alternativa è rappresentata dal Comal, anche se non è ancora disponibile ufficialmente. Il problema con questi due ultimi candidati è che non sono familiari al programmatore casuale. Il Comal è stato sempre disponibile per tutti i computer Commodore precedenti e per quelli MS-DOS, ma il numero di coloro che lo usavano era piccolo in confronto a quelli che gli preferivano il Basic.

ARexx (nella forma di Personal Rexx) è disponibile sulle macchine MS-DOS, ma, ancora una volta, sono in pochi coloro che lo utilizzano veramente.

Sebbene entrambi questi linguaggi siano abbastanza facili da apprendere, sono comunque linguaggi che devono ancora essere imparati, mentre la maggior parte dei precedenti possessori di home computer sono perlomeno familiari con il Basic.

Penso che la Commodore dovrebbe contattare una compagnia come la HiSoft con il proposito di ottenere una licenza per l'inclusione del suo pacchetto in ogni Amiga. Gli acquirenti della macchina gliene sarebbero estremamente grati.

Cosa ne dici, Commodore?

I messaggi di Amiga

di Don Curtis

Don Curtis è ufficiale di polizia a Denver, Colorado. Nell'ultimo paio di anni è stato assegnato alla progettazione e sviluppo di programmi insieme alla progettazione e alla gestione di sistema di un gruppo di 10 computer AT&T Unix 3B2. Nel suo tempo libero, Don è assistente Sysop nelle aree conferenze dedicate ad Amiga su CompuServe.

È strano come le discussioni su situazioni ipotetiche diventino improvvisamente reali. Ho avuto una breve discussione su CompuServe riguardo una delle cosiddette 'leggi di Murphy'.

La cosa è cominciata quando qualcuno ha detto di aver fritto la testina da 9000 dollari di una particolare marca di stampanti termiche. Aveva inavvertitamente tolto l'alimentazione e quindi l'aveva velocemente ripristinata. Ciao-ciao alla testina della stampante! Questa particolare stampante era stata prodotta da una compagnia rinomata per la robustezza e la longevità dei suoi prodotti, quindi è stato abbastanza sorprendente che fosse accaduta una cosa del genere.

La discussione si è spostata poi su come costruire circuiti di protezione e di ritardo per evitare che succedano queste cose e su quanto sia economico costruire uno di questi circuiti (specialmente quando si paragona al prezzo della testina della stampante). Questo, ovviamente, mi ha portato a ripetere una delle leggi di Murphy:

Il costo di un'apparecchiatura è inversamente proporzionale al costo della parte che si guasterà e che causerà la conseguente distruzione dell'apparecchiatura stessa.

In altre parole, un componente del costo di 5 dollari distruggerà un pezzo da 1000 dollari dell'apparecchiatura, mentre una componente da 1 dollaro coinvolgerà un pezzo da 50000 dollari.

Ma cos'ha tutto questo a che fare con Amiga? Ebbene, cari amici, leggete e lo scoprirete.

Quant'è sporco il vostro computer?

Ormai è quasi un anno e mezzo che posseggo il mio A2000 e lo uso per quattro o cinque ore al giorno. La mia stanza di lavoro è abbastanza piccola e si trova nel seminterrato, senza finestre o altre fonti di aria esterna. Il seminterrato non è riscaldato e nemmeno la mia computer room. Come molti seminterrati, la temperatura non diventa mai polare, ma scende di quel tanto che basta a impedire di lavorare confortevolmente. Per questo fatto, tengo sempre la stanza chiusa e quando devo lavorare uso una piccola stufetta elettrica per renderla più accogliente.

Il fatto che la stanza sia sempre chiusa non rappresenta un problema, eccetto per il fatto che io fumo. Ho provato svariati rimedi per catturare il fumo che produco e quello più semplice è di smettere di fumare, ma nonostante il fatto che vivrei sicuramente più a lungo, non sono riuscito a tenere l'aria della stanza completamente pulita. Questo implica che ogni anno o due, pulisco il mio computer per liberarmi dello sporco e della polvere che si sono accumulati.

Qualche notte fa, mi era sembrato che fosse arrivato il momento giusto per effettuare una pulizia, specialmente perché il mio hard-drive aveva cominciato a cigolare. Il rumore che sentivo era quello causato tipicamente dal blocco di eliminazione delle cariche statiche sul perno che cominciava a ballare. Questo non significa che ci sia qualcosa che non va, solamente dà abbastanza fastidio.

Prima che pensiate che mi sono preso quello che mi meritavo dal momento che fumo, lasciatemi dire che ho avuto hard-drive che producevano già questo rumore quando erano nuovi fiammanti, appena tirati fuori dall'imballo. Ho visto anche hard-drive tenuti in stanze apposite per computer, dove l'aria è filtrata e condizionata e non è permesso fumare, che hanno cominciato lo stesso a cigolare. Sono sicuro che il mio fumare non sia d'aiuto, ma non è neanche la causa del rumore. Succede a qualche drive.

Se non sapete cosa sia un blocco di eliminazione delle cariche statiche, vi diro che è un piccolo pezzo conduttivo tenuto contro la fine del perno da un pezzo di molla di rame che è saldata al piano di terra del drive. Questo impedisce che si formino delle cariche elettrostatiche sui dischi del drive e ciò è una buona cosa, quindi non vorrete certo asportare questo blocco, come ho visto fare a della gente.

Comunque, hard-drive e fumo a parte, è una buona idea tenere le cose pulite quando si tratta di computer, quindi mi sono preparato ad affrontare il lavoro.

A questo punto è opportuno dire due parole di avvertimento. Se non avete familiarità con le apparecchiature elettroniche, evitate di farlo da soli. È possibile danneggiare i chip con delle scariche elettrostatiche, potreste spostare delle parti fuori dal loro allineamento, usare il tipo sbagliato di solvente per pulire o molte altre cose che possono rovinare il vostro computer. Se non siete sicuri di quello che state facendo, per favore portate il computer da un rivenditore e fate eseguire le operazioni di pulizia dal loro tecnico.

Naturalmente vi faranno pagare per questo servizio, ma a lungo andare ne sarà valsa la pena per la tranquillità generata dal

sapere che il lavoro è stato eseguito correttamente.

La primissima cosa da fare se avete un hard-drive è fare il backup dei dati e parcheggiare le testine! Non importa quanta attenzione facciate, qualcosa può andare sempre storto (non è anche questa una legge di Murphy?), quindi assicuriamoci prima, di poter eventualmente recuperare le nostre cose se fosse necessario poi.

Dopo questa operazione, il primo passo consiste nello scollegare il cavo di alimentazione e tutti gli altri cavi connessi al computer. Ovviamente non vogliamo certo prenderci la scossa o tenere il computer acceso mentre lo puliamo.

Adesso togliamo la parte superiore del mobiletto. Yuch! Guardate tutta quella polvere accumulata vicino alla ventola, vicino ai connettori del mouse e della tastiera e ad ogni altra apertura sull'esterno. Duemila ore di aria risucchiata attraverso il computer produrranno sicuramente questo risultato. Non importa quanto sia pulita la vostra stanza, la polvere verrà risucchiata e si accumulerà attorno alla parte frontale del computer.

Non useremo un normale aspirapolvere per risucchiare lo sporco, perché questi ultimi generano un sacco di elettricità statica! Quei piccoli aspirapolvere che si tengono in una mano, chiamati anche 'aspirapolvere per computer', vanno bene per questo scopo, ma prima bisogna fare una piccola modifica: bagnate la piccola sacca con un po' d'acqua in modo che la polvere non passi attraverso.

Un altro modo per eliminare la polvere consiste nell'uso di una bomboletta di aria compressa. Potete procurarvene una in quasi tutti i negozi di materiale fotografico, ma *leggete* le istruzioni prima di usarla. Molte di queste bombolette di aria compressa diventano estremamente fredde se vengono usate per più di un secondo alla volta. Soffiate via la polvere dal computer. Io utilizzo un piccolo pennello per aiutarmi in questo compito.

... adesso gli hard-drive

Ho scollegato tutti i cavi dall'hard-drive e dai floppy drive. Un piccolo trucco che può aiutarvi a riassemblare il tutto consiste nel marcare i cavi segnalando dove devono andare e quale parte va rivolta verso l'alto. I cavi dei drive (tipo piattina) hanno, normalmente, uno spazio sul lato superiore del connettore dove possiamo appiccicare una piccola striscia di nastro sul quale scrivere. Io uso questo metodo per annotare il numero del drive e qual'è la parte che guarda verso l'alto. Ho anche scollegato il connettore di alimentazione dalla piastra madre. Bisogna usare un piccolo cacciavite per allontanare il connettore dal gancio di sicurezza e bisogna fare attenzione a non tirare i fili perché potrebbero rompersi. Ho svitato quindi le viti che assicurano anteriormente e posteriormente il blocco alimentatore/drive allo chassis e quindi l'ho sollevato e portato fuori interamente.

Successivamente ho smontato la piastra di fissaggio dei drive togliendo le quattro viti laterali. Ho svitato anche le quattro viti che fissavano il mio floppy drive da 5.25" e quindi l'ho estratto dal suo alloggiamento.

Come la parte frontale del computer, anche i drive erano ricoperti di polvere. Per pulirli è meglio utilizzare un piccolo pennello anziché un aspirapolvere o l'aria compressa. Se si usa un pennello con attenzione ci sono meno probabilità di arrecare danni. Nel mio caso, sulla piastra di fissaggio dei drive si trovano un floppy e un hard-drive da 3.5", quindi ho svitato le quattro viti sul fondo della piastra che tenevano ancorato lo hard-drive e l'ho tolto. Non ho smontato anche il floppy da 3.5" perché non lo ritenevo necessario. Smontando l'hard-drive, invece, si può accedere al blocco di eliminazione delle cariche statiche. Per favore, fate estrema attenzione nel maneggiare l'hard-drive. Sebbene gli ultimi modelli siano piuttosto robusti se paragonati a quelli disponibili pochi anni fa, sono ancora abbastanza delicati in paragone ad altre apparecchiature.

Girate l'hard-drive sulla schiena e vedrete sulla piastra di controllo una piccola linguetta di rame sovrastante un piccolo foro nel circuito stampato. Su questa linguetta, in genere, è fissato un piccolo blocco (normalmente nero) che viene tenuto premuto contro il perno. Dovrebbe essere abbastanza facile da identificare. Una volta trovato, lo pulisco con del solvente per la pulizia dei contatti elettronici (per esempio lo Zero Residue Cleaner della Tandy o qualsiasi analogo prodotto in vendita nei negozi di materiale elettronico). La marca non ha importanza, assicuratevi solo che il prodotto che intendete usare *non* lasci alcun residuo e che il suo uso sia previsto per le parti elettroniche. Non usate *mai* un prodotto a base di acqua, nè alcun prodotto non espressamente indicato per apparecchiature elettroniche. Io spruzzo questo prodotto anche sul circuito stampato per eliminare gli eventuali film di sporco che dovessero essersi formati. Non ne serve molto, quindi fate attenzione a non spruzzarne troppo e a non farlo andare sulle parti meccaniche del drive.

Sebbene questa procedura sia normalmente sufficiente a pulirlo e a rendere il suo funzionamento nuovamente silenzioso, io vado ancora più in là; il prossimo passo *non* è raccomandato dai produttori di hard-drive. Io l'ho fatto per anni e non ho mai avuto problemi, ma questo non significa che non li avrete neppure voi. Se intendete provarlo, ricordate che lo fate a vostro richio e pericolo!

Il passo extra che io intraprendo consiste nel mettere una piccola goccia di olio chiaro sul blocco. La parola importante qui è *piccola*. Verso una goccia di olio dal beccuccio di un piccolo contenitore e quindi la sfioro immediatamente con un fazzoletto di carta per assorbire l'eccesso. Questa operazione ha sempre funzionato bene per quanto mi riguarda e ha reso silenziosi drive che non ero riuscito a zittire in altro modo, ma ricordate ancora una volta che la fate a vostro richio. Ho effettuato poi la stessa procedura con l'hard-drive collegato alla bridge-board (Janus).

... e adesso i floppy

Sul mio floppy da 3.5" (Matsushita) ci sono due piccole viti che fissano uno schermo metallico. Togliendo queste viti e rimuovendo lo schermo, potete raggiungere le testine abbastanza facilmente. Questa operazione rende anche facile il compito di

spazzolare via lo sporco accumulatosi all'interno del drive. Dopo questo, spruzzo un po' dello spray visto prima per lo hard-drive su un bastoncino cotonato e lo strofino attentamente sulle testine (ce ne sono due, una sopra e una sotto) da parte a parte. Non strofinate il bastoncino in senso avanti-indietro, in modo da non dovervi preoccupare di spingere le testine fuori allineamento. Le probabilità che ciò avvenga sono molto limitate, ma perché rischiare? Strofinare in senso laterale funziona altrettanto bene e non presenta pericoli.

La pulizia del drive da 5.25" (il mio è un Chinon) è ancora più semplice. Non bisogna rimuovere alcuno schermo per arrivare alle testine. Basta applicare la stessa procedura, pulire la polvere con un pennello e strofinare le testine da parte a parte con il bastoncino cotonato imbevuto del solito solvente. Io mi spingo fino a rimuovere la mascherina frontale e a pulirla con un detersivo casalingo, ma questo è un'operazione puramente cosmetica e non ha effetto sul buon funzionamento del drive. Spero che sia chiaro che dovete prima rimuovere la mascherina frontale se volete pulirla con una soluzione a base di acqua e detersivo. Se lo fate, assicuratevi che sia completamente asciutta prima di rimontarla al suo posto.

... adesso, il circuito stampato

La prossima e ultima operazione all'interno del computer consiste nel rimuovere tutte le schede di espansione per pulire i loro contatti. Per fare questo basta prendere un foglio di carta da cucina, spruzzarvi il solito solvente per la pulizia dei contatti elettronici e strofinare questo foglio sui contatti a pettine della scheda. A parte questa operazione, non è necessario fare altro sulle schede di espansione, eccettuata la solita spennellata per rimuovere la polvere.

A questo punto abbiamo a disposizione la scheda madre nuda e cruda. Senza schede e connettori non è difficile spazzolare la polvere che si è accumulata anche su di essa. Fatto questo, controllo che tutti i chip zoccolati siano ben inseriti nella loro sede. È sufficiente una breve pressione verso il basso applicata sul loro dorso. Se non erano ben inseriti, li sentirete spostarsi verso il basso. Se erano già ben inseriti, non spingete troppo perché potreste rovinare la scheda madre.

... rimettiamo tutto a posto

Rimontate il drive da 5.25" nella sua sede e l'hard-drive da 3.5" sulla sua piastra di supporto. Assicuratevi di posizionarlo indietro quanto basta per non urtare il pannello frontale dello chassis.

Come passo successivo rimontate la piastra di supporto dei drive sul blocco di alimentazione e quindi rimettete il tutto nella sua sede naturale all'interno del computer. Assicuratevi che le linguette metalliche del blocco di alimentazione siano nelle posizione corrette dello chassis e poi riavvitate le viti di fissaggio. Sul mio A2000 le due viti frontali non avevano la rosetta, mentre le quattro posteriori ne erano provviste. Non so se questo sia il caso di tutti, ma se lo è, sapete dove mettere quelle con la rosetta e quelle senza.

Adesso rimettete le schede nei loro slot. Non posso rendere l'idea di quanto sia importante che le schede vengano inserite fermamente fino in fondo allo slot. Una scheda come la BridgeBoard vi darà sicuramente dei problemi se non sarà propriamente inserita. Ricordate, inoltre, che la staffa può rimanere incastrata e sollevata sul fermo che si trova sotto la scheda madre. Come ho già avuto occasione di dire in un mio precedente articolo, io piego leggermente all'infuori la parte inferiore della staffa per rendere l'installazione della scheda notevolmente più facile.

Il prossimo passo consiste nel ricollegare tutti i cavi. Assicuratevi che questi siano correttamente collegati agli svariati drive. Se avete seguito il mio consiglio di contrassegnarli, l'operazione dovrebbe risultare abbastanza semplice. Il cavo di alimentazione della scheda madre scatta in posizione di blocco quando viene inserito correttamente. I cavi di alimentazione per i drive sono costruiti in modo da non poter essere inseriti in maniera errata (questo vale per i connettori di tipo grosso) tranne che per i connettori di tipo piccolo (floppy da 3.5") con i quali non è possibile sbagliare il verso d'inserimento ma è pur sempre possibile inserirli sfasati di lato. Fate particolare attenzione a questo.

Okay, controllate il vostro lavoro. Sembra tutto a posto? Se sì, proseguite ricollegando tutti i cavi esterni a eccezione del cavo di alimentazione. Questo cavo sarà l'ultima cosa che ricollegheremo. Potete lasciare aperto il computer, in caso ci sia qualcosa che non va.

... adesso passiamo alla tastiera

Dovete prestare *molta* attenzione a questo punto. Potete pulire i tasti sia mentre sono montati sulla tastiera, sia dopo averli staccati da essa. Vi raccomando vivamente di staccare i tasti solo se manifestano problemi di qualche tipo (un tasto che rimane premuto o che non funziona correttamente). Se li pulite mentre sono sulla tastiera, usate un panno inumidito con del detergente. Non usate un panno *troppo* umido perché potrebbe gocciolare dentro la tastiera. Potete usare un bastoncino cotonato bagnato di detergente per raggiungere la parti più inaccessibili.

Se avete dei tasti che tendono a incastrarsi (a rimanere appiccicati sul fondo) o che non rispondono correttamente quando li utilizzate, dovrete rimuovere il cappuccio. È molto importante fare la massima attenzione perché potete rovinare una tastiera se non tirate via i cappucci nella maniera giusta! Questi cappucci sono tenuti in posizione da linguette a scatto su entrambi i lati del tasto. In molti negozi di computer potrete comperare un apposito attrezzo per smontare i tasti. In alternativa si può usare qualcosa simile a un uncino per bottoni prendendo un lato del cappuccio e tirandolo gentilmente verso l'alto e contemporaneamente verso l'esterno. Il cappuccio verrà via improvvisamente. I tasti più grandi, come la barra dello spazio, sono tenuti in posizione anche da bracci metallici. Possono essere molto difficili da reinserire, quindi è meglio lasciarli perdere.

Quando togliete i cappucci dai tasti, quello che vedrete dipende dal produttore della tastiera. La mia, in particolare, è stata pro-

dotta dalla HiTek e ha dei piccoli interruttori a foglia che vengono tenuti aperti dal meccanismo a scorrimento. Spingendo in giù il meccanismo, l'interruttore si chiude per poi riaprirsi quando il meccanismo torna verso l'alto. Questi interruttori a foglia sono molto delicati, quindi statene alla larga! Una piccola spruzzata del detergente già usato per l'hard-drive dovrebbe risolvere i problemi di tasti appiccicosi o malfunzionanti.

Se estraete accidentalmente il meccanismo di scorrimento, non fatevi prendere dal panico. Sono abbastanza facili da rimettere dentro, se fate abbastanza attenzione. Come ho detto, l'interruttore a foglia è tenuto lontano dai contatti dal meccanismo di scorrimento e se questo viene fuori, non si può semplicemente rimetterlo dentro. Prendete qualcosa di simile a uno stuzzicadenti e mettetelo fra le foglie, sollevandolo leggermente per separarle. Mettete la parte scorrevole sulla sua molla e spingetela lentamente verso l'interruttore, facendo attenzione a non prendere le foglie dell'interruttore. Una volta portato questo pezzo sopra l'interruttore, scatterà nella propria posizione sulla piastra sottostante.

Rimettere i cappucci è un'operazione semplice. Basta appoggiarli sui tasti con l'orientamento giusto ed esercitare una leggera pressione per farli scattare nella propria sede.

... facciamo il bagno al mouse

L'unica cosa che faccio al mouse è quella di pulire la sfera e le rotelline situate all'interno della carcassa. Se girate il mouse, vedrete la piastrina rimovibile per accedere alla sfera. Ruotatela nel senso indicato dalla freccia o consultate il vostro manuale che spiega le operazioni da compiere per rimuoverla. La sfera è facile da pulire, con un po' d'acqua e sapone. Assicuratevi solo che sia completamente asciutta prima di rimetterla nel mouse. Mentre è fuori, controllate le rotelline all'interno del mouse. Dovrebbero essere pulite e luccicanti.

Non c'è un piccolo anello di gomma al centro di queste rotelline, ma, se vedete qualcosa che gli somiglia, ebbene questo è sporco e dovrebbe essere rimosso. Se avete una lente d'ingrandimento, date un'occhiata 'all'anello di gomma' e quindi grattatelo gentilmente via. Potete usare uno stuzzicadenti di legno per eseguire quest'ultima operazione. Una volta ripulita la sfera e le rotelline, rimettete la prima al suo posto e richiudete la piastrina.

... e infine, il monitor

Ricollegate il mouse e la tastiera al computer. Adesso rimane da pulire solo lo schermo del monitor. Ci sono cose che *non* dovete *mai* fare attorno a un monitor! Un monitor (a colori o monocromatico) utilizza alti voltaggi, fino a 30000 volt, per pilotare il tubo catodico e questi potrebbero UCCIDERVI!

Non pulite un monitor mentre è acceso. *Non* usate uno straccio gocciolante per pulire un monitor. *Non* spruzzate mai uno spray direttamente sullo schermo, ma spruzzatelo su uno straccio e quindi usate quest'ultimo per pulire lo schermo. *Non* aprite mai la custodia del monitor o pasticciate al suo interno. Lasciate queste cose a una persona qualificata, che abbia gli attrezzi e la conoscenza adatta a lavorare su questi voltaggi.

Per pulire lo schermo del monitor, spruzzate uno straccio con del liquido di pulizia per i vetri o con un detergente casalingo e strofinate lo schermo fino a quando non lo avrete pulito. Ancora una volta, per favore fate attenzione a non far entrare del liquido all'interno del monitor. Lo schermo si pulirà bene anche con piccole quantità di detergente.

Okay, adesso è tutto pulito! Assicuratevi che tutto sia asciutto e ricollegate il monitor al computer. Controllate tutte le connessioni. Se non avete ancora rimesso il coperchio al computer, ricontrollate tutti i cavi. Infine, reinserite il cavo di alimentazione di rete nel computer e accendetelo. Se succede qualcosa di anormale, spegnete il computer immediatamente!

Murphy colpisce ancora

Quando ho fatto questa operazione... aargh!!! Il computer non si è acceso! Il LED di alimentazione non si è illuminato, la ventola dell'alimentatore non è partita e i drive non hanno cominciato a girare! Cosa avevo sbagliato?

Ho ricontrollato tutti i connettori ma non ho visto nulla di immediatamente ovvio. La spina era collegata alla presa a muro e la stessa era sotto tensione.

Ebbene, ho scollegato il cavo di alimentazione da tutti i drive e ho riprovato ad accendere il computer. La ventola ha cominciato a girare e il computer ha iniziato le operazioni di boot. Senza alimentazione ai drive non potevo fare niente, ma mi sentivo incoraggiato. Ho spento di nuovo tutto e quindi ho collegato l'alimentazione dell'hard-drive. Ho riacceso il computer e tutto funzionava correttamente, ma il floppy era ancora senza alimentazione quindi non potevo eseguire il boot.

Ho spento di nuovo tutto e quindi ho raggiunto la parte posteriore del floppy tentando di inserire il cavo di alimentazione nel suo connettore, rendendomi conto che veniva accettato in varie posizioni diverse! A causa della disposizione della mia computer room non potevo girare facilmente attorno al computer per guardare il floppy da dietro e il connettore di alimentazione non è nemmeno visibile dall'alto. Ho afferrato uno specchio e ho guardato la connessione, scoprendo cosa avevo sbagliato. Senza fermi laterali sul connettore, non c'è niente che impedisca di inserire il cavo in posizione sbagliata lateralmente nel connettore! Non potete accoppiare cavo e connettore invertendo il sopra col sotto, ma nulla vi impedisce di spostare il cavo di uno o due pin sulla destra o sulla sinistra! Questo voleva dire che avevo messo il filo dei 12v o dei 5v direttamente a massa! Nessuna sorpresa, quindi, che il computer non partisse. Ho collegato correttamente questo connettore e ho riacceso il computer... tutto ha funzionato alla perfezione.

E qui entra in ballo la legge di Murphy. Per la mancanza di un pezzetto di plastica, del valore di un decimo di centesimo di dollaro, da mettere ai lati del connettore, avrei potuto rovinare un computer del valore di migliaia di dollari. Questo non è colpa della Commodore perché loro comprano i drive da svariati produttori che li forniscono esattamente così. È colpa degli in-

gegneri che hanno progettato questa combinazione di connettori maschio-femmina 'standard' da utilizzare per i drive di piccole dimensioni. È anche stata colpa mia per non avere utilizzato lo specchio in precedenza, per vedere cosa stavo facendo. Avevo inserito il connettore fidandomi solo del tatto e credevo di averlo messo bene... ovviamente non era così.

Vi serva da lezione. Non importa quanti computer abbiate smontato in passato, non importa quanto siate familiari con le apparecchiature elettroniche... prendetevi il tempo necessario per assicurarvi che abbiate fatto tutto correttamente. Come dice la prima legge di Murphy, "Se qualcosa può andare storto, lo farà!".

L'operazione di pulizia è valsa la pena. Ha richiesto solo poche ore del mio tempo e l'hard-drive non cigola più. La tastiera è tornata in perfetta efficenza e il monitor ha tutto un altro aspetto senza quello strato di sporco sullo schermo. Non mi devo più preoccupare per l'accumulo di polvere all'interno dei drive e di tutto il resto. Il computer sembra come nuovo.

Riparazioni della BridgeBoard

Non ho mai avuto l'intenzione di divenatare un 'guru' della BridgeBoard ma sembra che i problemi e/o le novità continuano a emergere tutto il tempo, quindi ecco un aggiornamento sulla situazione.

La mia BridgeBoard è defunta una settimana fa. Si comportava stranamente già da un bel po' di tempo. Sembrava che avesse una sua mente capricciosa per quanto riguardava il boot. Qualche volta effettuava questa operazione correttamente, qualche altra bloccava la macchina o la mandava in Guru. Sembrava anche molto sensibile a come veniva esattamente posizionata nello slot di espansione. Qualsiasi deviazione dalla posizione *giusta* ne causava l'immediata cessazione dell'attività.

Ebbene, la settimana scorsa ha deciso che era ora di andarsene definitivamente... si rifutava di funzionare, qualsiasi cosa facessi. Non riconosceva i drive e il comando *BindDrivers* si bloccava, fermando l'intero Amiga o mandandolo a meditare.

Per quelli di voi che stanno pensando che devo avere fatto qualcosa di dannoso mentre pulivo il computer, posso dire che non è assolutamente possibile, perché la scheda ha funzionato bene per un bel pezzo dopo quella operazione. In realtà, sembrava che funzionasse addirittura meglio del normale, in quanto non mi ha più dato alcun problema ... fino a questo.

In ogni caso, ho provato tutti i trucchi che conoscevo per riportarla alla ragione, come risistemare i chip negli zoccoli, le schede nei connettori, ricaricare il software e così via. Non ha funzionato niente, quindi sono andato dal mio rivenditore per farla riparare.

E qui è dove la cose hanno cominciato a farsi interessanti. Il tecnico di servizio mi ha detto che la CBM gli aveva fornito un disco diagnostico per la BridgeBoard, ma che per usare questo disco bisognava prima far partire con successo la scheda. Stava parlando del programma Catch-22. Non potete diagnosticare il problema se non riuscite a eseguire il boot della Bridge, ma se la Bridge è rotta, non esegue il boot. La risposta della Commodore a questo problema consiste nella sostituzione dell'intera scheda per 235 dollari!

Roger (il tecnico) e io siamo abbastanza svegli da sapere che la CBM deve avere qualche metodo per testare le BridgeBoard. Dopo avere fatto svariate telefonate alla CBM, Roger è riuscito a sapere che in effetti tale metodo hardware/software è disponibile in *Europa* ma non qui negli US. Sembra che gli utenti europei non siano disposti a subire la sostituzione dell'intera scheda e che preferiscano l'individuazione del guasto sulla loro scheda e la relativa riparazione. La Commodore europea ha quindi provveduto a fornire quello che il mercato richiedeva. I dirigenti Commodore qui negli US non sono della stessa idea e pensano che a noi vada bene la sostituzione integrale della scheda anziché fornire ai laboratori gli strumenti per riparare le schede difettose.

I dipendenti della Commodore di qui ci hanno detto che la scheda non è 'completamente a posto' (ha dei bug) ma che prima o poi in futuro avrebbero messo in distribuzione il materiale diagnostico per la scheda, quando questi programmi e l'hardware fossero stati messi 'a posto'.

Lasciatemi fare una digressione dallo specifico soggetto della riparazione della BridgeBoard alle riparazioni in generale. In alcuni casi la sostituzione completa di una scheda è la soluzione migliore al problema, in altri non lo affatto. Il costo del servizio di assistenza varia fra 40 e 60 dollari all'ora, a seconda di dove abitate e di quanto è in gamba il tecnico. Alcuni problemi, soprattutto quelli intermittenti, sono estremamente difficili da diagnosticare. La maggior parte dei chip del computer sono saldati al loro posto. La saldatura dei chip direttamente alla scheda consente di abbassare i costi di produzione e aumenta l'affidabilità generale del sistema. Non bisogna preoccuparsi di chip non bene inseriti nei loro zoccoli, però bisogna preoccuparsi dei costi di riparazione.

Un esempio dei problemi di riparazione (e delle relative spese) che questo fatto può causare è la situazione nella quale si tenta di seguire un certo segnale. Il segnale entra correttamente nel chip A, ma sparisce all'uscita del chip A in direzione del chip B. Questo può voler dire che il chip A è guasto in quanto ha l'uscita rotta, o che il chip B è guasto perché ha l'ingresso in corto-circuito. Con i chip su zoccolo si sostituisce semplicemente il chip B e, se il segnale ritorna a funzionare correttamente, allora è proprio B il chip guasto. In caso contrario significa che è guasto il chip A.

Con i chip saldati in posizione, non è così semplice. Con le schede a più strati bisogna fare molta attenzione quando si dissalda un chip. È abbastanza facile distruggere una scheda a più strati con tecniche di saldatura non adatte. È anche abbastanza comune il fatto di distruggere il chip che si sta dissaldando. Se si tratta di un chip da 3 dollari non è un grosso problema, ma con uno che costa 50 dollari o più il problema è grosso.

Il tecnico, quindi, deve tirare a indovinare su quale sia (A o B) il chip guasto e sostituirlo. Se il tecnico si è sbagliato deve so-

stituire anche l'altro chip. Tutto questo richiede tempo e soldi, oltre ai costi delle parti di ricambio, che possono essere decisamente alti. In casi come questi, dove non è possibile isolare e risolvere rapidamente il problema, la sostituzione dell'intera scheda è probabilmente una buona idea. A lungo termine, risulta meno costoso per il cliente e il tecnico non deve perdere tre o quattro ore per isolare il problema su una macchina, trovandosi così molto presto con una pila di macchine che si accumulano in attesa della riparazione. Questo non farebbe certo piacere ai clienti che dovrebbero aspettare una o due settimane prima che il tecnico possa prendere anche solo in considerazione la loro macchina.

Ci sono, a ogni modo, casi in cui la sostituzione dell'intera scheda è ridicola. Uno dei membri di CompuServe segnalò che il suo A1000 si era guastato. Lo portò a riparare e gli dissero che aveva bisogno di una nuova scheda madre. Questo gli costò 180 dollari più il lavoro per sostituirla. Quando chiese cosa ci fosse di rotto nella vecchia scheda madre, gli venne risposto che uno dei chip di memoria era guasto. Se questa non era una giustificazione qualsiasi con la quale i negozi accontentano i clienti quando chiedono qual'era il guasto, allora la spesa di 200 dollari o più perché si era guastato un chip da 10 dollari è assurdo! *Se* erano stati in grado di diagnosticare che si era guastato un chip di memoria, avrebbero automaticamente dovuto essere in grado di isolare e sostituire quel chip a una frazione del costo di 200 dollari.

Ma, siccome la Commodore non fornisce i mezzi di diagnostica necessari, molti negozi non assumono personale di riparazione qualificato. Assumono, invece, della gente che ha delle capacità di diagnostica minime ma che sono in grado di sostituire chip zoccolati o schede intere. Si sono così trasformati da laboratori di riparazione a laboratori di sostituzione. I loro cosiddetti *tecnici*, eseguono quei minimi controlli sui chip che possono sostituire e se questi non risultano malfunzionanti, mandano la scheda indietro per essere rimpiazzata.

Se la Commodore riuscirà mai a *sistemare* i suoi programmi diagnostici, allora forse avremo nuovamente dei laboratori di riparazione. Laboratori nei quali il tecnico sappia quali problemi richiedono la sostituzione completa della scheda e quali invece possono essere risolti abbastanza facilmente in loco.

Penso che, adesso come adesso, la Commodore stia offrendo alla comunità di Amiga un vero e proprio disservizio, non fornendo ai tecnici gli strumenti adeguati per lavorare. Anche strumenti non perfettamente funzionanti sarebbero meglio, a mio parere, che nessuno strumento. Fino a quando sono conosciuti i limiti dell'attuale software diagnostico, un tecnico qualificato può sempre cavarsela aggirando questi limiti con le sue conoscenze.

Ad ogni modo, questi strumenti non sono disponibili qui negli US, quindi torniamo al mio problema con la BridgeBoard.

Prima di decidere di mandare indietro la mia Bridge perché fosse sostituita, Roger decise di provare a isolare il problema. Aveva una Bridge perfettamente funzionante in negozio e dalle sue chiamate alla Commodore era stato in grado di ricavare degli indizi sul possibile guasto. Perlomeno poteva provare a cambiare i chip zoccolati con quelli della scheda funzionante per vedere se era possibile isolare il problema in uno di questi.

Cominciò a sostituirli uno alla volta, fino a quando la scheda si comportò come se stesse per rimettersi a funzionare. Ogni volta che sfilava un vecchio chip, piegava leggermente in fuori i pin dello zoccolo per assicurare un buon contatto al nuovo chip che sarebbe stato inserito (questa operazione veniva effettuata per i larghi chip quadrati). Naturalmente, durante questo lavoro, dovette risistemare più di una volta la scheda nel connettore, perché questa scheda, non importa cosa si faccia, deve essere inserita correttamente e non è delle più facili da posizionare.

Quello che succedeva ora era che la scheda accedeva al drive quando raggiungeva il comando *BindDrivers*, ma questo non voleva saperne di resettarsi. Roger disse che aveva visto questo problema sui primi drive da 5.25" che la Commodore aveva fornito con le Bridge. Qualche volta si bloccano sulla traccia 0 e non si spostano o resettano, anche spegnendo e riaccendendo il drive. Aveva trovato un rimedio molto semplice per uscire da questo impasse. Bastava spegnere il drive e muoverne la testina un poco verso l'esterno, in modo da toglierla dalla posizione di parcheggio. Sembra che questa operazione sia sufficiente a far sì che il drive si *svegli* la prossima volta che viene acceso. Ad ogni buon conto, questo si può fare con un floppy drive da 5.25", *non* con un hard-drive!

Ebbene, che io sia dannato se non ha funzionato. Riaccendendo Amiga, la Bridge effettuò correttamente il boot, adesso che il drive si ricordava di essere vivo. Tornando al Catch-22, non c'era alcuna necessità di lanciare il programma diagnostico adesso: la scheda funzionava perfettamente. Adesso dovevamo soltanto rimettere dentro i vecchi chip, uno alla volta, per vedere quando si sarebbe guastata di nuovo e sperare che quello fosse il chip che andava sostituito.

Yup, avete indovinato. Sostituimmo tutti i chip originali e la scheda continuò a funzionare perfettamente. Apparentemente il problema era dovuto a uno dei pin nello zoccolo di uno dei chip che si era piegato (probabilmente il Bus Translator). Non faceva un buon contatto e la normale ossidazione del metallo aveva fatto sì che smettesse completamente di fare contatto elettrico. Un'ora di lavoro e la mia Bridge era riparata. L'ho portata a casa, rimessa nel mio Amiga e da allora funziona bene. In realtà funziona anche meglio di prima. Non sono più necessari i trucchi di posizionamento. È scivolata nel suo connettore e ha cominciato a funzionare al primo colpo, anche se ho dovuto usare il trucco di muovere la testina del mio drive da 5.25" la prima volta che l'ho accesa.

Da allora, comunque, ha funzionato bene. Questo della testina del drive bloccata non era il problema originale. Anche se la testine erano bloccate, il LED del drive si accendeva e la Bridge effettuava il boot fino al punto in cui cercava il DOS sul disco nel drive A:. Quando avevo portato la mia Bridge a riparare, non passava neanche oltre al comando *BindDrivers* e il LED del drive non si accendeva per niente. Ho il sospetto che quando la mia Bridge non funzionava, in un modo o nel-

l'altro è riuscita a bloccare le testine del drive nella loro posizione. Non so se questo sia veramente possibile, ma è interessante il fatto che sia successo su due macchine quando su di esse è stata usata la mia Bridge difettosa.

Alla fine ho pagato un quinto di quanto mi sarebbe costato il cambio dell'intera scheda. Il morale di questa storia è che se il vostro Amiga dovesse guastarsi, cercate di trovare un negozio dove saranno disposti almeno a tentare qualcosa prima di decidere che dovete spendere svariate centinaia di dollari per cambiare qualche scheda. Un tecnico competente può normalmente dire, abbastanza velocemente, se la via più economica consiste nella sostituzione o nella riparazione della scheda. Un buon tecnico potrà anche effettuare quelle piccole operazioni come la risistemazione dei pin degli zoccoli, giusto per essere sicuri che il problema risieda nel chip e non nello zoccolo (come è successo nel mio caso).

Potrebbe essere utile andare dal vostro rivenditore e fare una chiaccherata con il tecnico. Anche se il vostro Amiga funziona bene, potrete perlomeno farvi qualche idea su come lavori quel particolare negozio. Se è solamente un negozio *di sostituzione*, potreste voler guardarvi in giro per trovare un altro centro di assistenza. Si spera che il vostro Amiga non si rompa mai, ma se lo dovesse fare, potrete risparmiare tempo e soldi se saprete già a chi rivolgervi per le riparazioni. E mentre vi guardate intorno, potreste vedere se il tecnico riesce a convincere la Commodore a distribuire il software diagnostico che hanno in questo momento.

Nuovi giocattoli

Mentre ero dal mio rivenditore, ho avuto l'opportunità di dare un'occhiata alla nuova Bridge Board A2286 AT-compatibile. Ne avevano ricevute tre il giorno prima, le prime tre arrivate a Denver.

È una bella scheda, ma abbastanza costosa al prezzo di listino di 1499 dollari. La A2286 è un clone AT-compatibile a 8 MHz, 1 Mbyte di RAM costituita da una scheda principale e da una daughter-board. Occupa due slot di Amiga, uno dei quali deve essere uno slot Bridge. Questo vuole dire che se la mettete nello slot di sinistra, non potrete più utilizzare l'altro slot AT-compatibile. Se lo desiderate, comunque, potrete sempre installare i due connettori da 36 pin per rendere tutti gli slot MS-DOS compatibili con gli slot AT. Sulla scheda madre dell'A2000 sono già presenti i fori e le piste; basta semplicemente saldare i connettori al loro posto. Se mettete la A2286 nel connettore Bridge di destra, occuperete anche lo spazio di uno dei connettori Amiga, mantenendo però libero l'altro slot Bridge senza dover modificare la scheda madre.

La A2286 viene fornita con un floppy 5.25" da 1.2 Mbyte, MS-DOS 3.3 e il nuovo software JANUS versione 2.0. Supporta i modi grafici MDA e CGA sul monitor di Amiga (come la Bridge A2088). Sulla scheda c'è anche lo zoccolo per il chip matematico 80287 nel caso desideraste utilizzarlo.

La scheda sembra ben costruita (non ci sono le modifiche con fili dell'ultimo minuto) ma c'è una cosa che manca clamorosamente ai miei occhi. Non c'è più il connettore per il drive esterno che la A2088 invece possiede! Se volete aggiungere un secondo floppy, dovrete procurarvi un drive, un alimentatore, una custodia e quindi fare passare la piattina dal retro dentro il buco lasciato aperto in corrispondenza di uno degli slot di Amiga. Trovare uno slot libero in Amiga e rimuoverne la relativa chiusura metallica non è un problema, perché si può usare quello immediatamente a destra della A2286. Quello slot non verrà mai utlizzato perché il suo spazio è occupato dalla daughter-board della A2286, ma per essere onesti, il tutto assume un'aria poco dignitosa. Inoltre, dal momento che sulla A2286 non c'è un connettore db23, non potete usare direttamente un A1020 (il drive da 5.25" della Commodore).

Sfortunatamente il mio rivenditore non aveva ancora installato la scheda in una macchina per dimostrarne il funzionamento, quindi non ne ho potuto verificare la funzionalità. Ho potuto, comunque, provare la nuova release 2.0 del software JANUS, perché lo stesso software funziona con la A2088 e con il SideCar. Tutto il software è completamente cambiato e sono state aggiunte alcune cose nuove.

Adesso si può usare il clock di A2000 per modificare la data e l'ora della parte MS-DOS. Potete usare il mouse di Amiga come un mouse MicroSoft per il bus. Si può effettuare il boot automatico della Bridge da un file AmigaDOS usato come drive reale (opposto a quello virtuale). In realtà, questo drive è così *reale* per la parte Bridge, che bisogna usare FDISK per partizionarlo e FORMAT per formattarlo prima di potervi accedere.

Non solo questo, ma quando lo creiamo, usando il programma Amiga MAKEAB, viene creato con le appropriate dimensioni massime. JLINK (che è ancora disponibile) non creava il suo drive virtuale con le dimensioni massime quindi noi dovevamo prima riempirlo per evitare il formarsi di errori nel caso ci fossimo dimenticati di effettuare l'unlink del drive (come abbiamo già visto nel primo articolo di questa serie).

AREAD e AWRITE hanno delle opzioni aggiuntive oltre allo switch /b e, questa volta, sono documentate. Normalmente, AREAD e AWRITE effettuano, durante le loro operazioni, sia conversioni di caratteri che la conversione di line-feed in carriage-return/line-feed. I nuovi switch ci permettono di disattivare l'una o l'altra conversione.

E dopo tutto quello che abbiamo dovuto fare un po' di tempo fa per montare il FFS su un drive JANUS, DJMount adesso supporta pienamente l'FFS. C'è anche un nuovo comando Amiga FORMAT che funziona su un drive JANUS sotto OFS o FFS. Già che siamo in argomento, vorrei segnalarvi un altro piccolo problema che è emerso usando il vecchio software JANUS insieme all'FFS su un drive JANUS. Alcuni lettori l'hanno provato usando la prima partizione sull'hard-drive MS-DOS come drive JANUS e non ha funzionato per niente. Hanno allora spostato la parte JANUS sulla seconda partizione e tutto ha funzionato bene.

Un'altra bella cosa del nuovo software è che se la Bridge non riesce a effettuare il boot, non riceverete più il requester 'Task Held' seguito dal Guru.

Alcune delle directory nella parte Amiga sono state spostate e quindi non potete mischiare le due versioni del software. Se intendete aggiornare il vostro software, pertanto, assicuratevi di sostituire tutti i file necessari e che questi vadano a finire al posto giusto che loro compete. Assicuratevi, inoltre, di lanciare PcPrefs e di stabilire la locazione in RAM per la JANUS prima di tentare di utilizzare la Bridge. Per il SideCar e la A2088 la giusta locazione RAM è $E000, mentre la A2286 usa la locazione $D000.

Fino a questo momento, la Commodore non ha fornito alcuna indicazione su come passare dalla vecchia versione del software alla nuova release 2.0. Non si sa se sarà permesso ai rivenditori di fare delle copie dei dischi con il nuovo software per quelle persone che hanno ancora quello vecchio, o se queste ultime dovranno acquistarlo come pacchetto a sé stante. Se volete aggiornarvi, rivolgetevi al vostro rivenditore... nel momento in cui leggerete questo articolo potrebbero avere deciso cosa fare.

Stanno arrivando altri giocattoli

AMIX (Amiga UNIX (tm)) e A2500ux sono stati descritti in un bollettino informativo (Technology Preview) della Commodore. Tali bollettini *non* annunciano la disponibilità del prodotto, né contengono le specifiche della versione finale del prodotto. Sono invece delle semplici comunicazioni sullo stato dello sviluppo di un certo prodotto e su come si pensa che verrà reso disponibile in un futuro non meglio precisato. Il bollettino su AMIX e A2500ux dice:

A2500ux verrà fornito con 1 Mbyte di RAM a 16 bit e 4 Mbyte di RAM a 32 bit sulla scheda A2620. Verrà incluso un harddrive da 80 Mega con il controller A2090a e un'unità di backup su nastro a cartucce da 150 Mbyte.

AMIX sarà completamente compatibile con la release 3.1 dell'AT&T UNIX System V. Sarà in grado di indirizzare 1 *Gigabyte* di spazio di indirizzamento virtuale con la tecnica della gestione a pagine della memoria. Verrà incluso il supporto software per più utenti, più schermi video, stream e un filesystem remoto in rete (RFS). Sarà dotato anche di un sistema propietario di Windowing.

Il software incluso con AMIX consisterà di un compilatore C ottimizzante con i tool standard per lo sviluppo come *cc*, *sdb*, *make*, *yacc* e *lint*. Verranno anche inclusi i programmi di utilità standard per la gestione di testo come *email*, *uucp*, *nroff*, *vi*, *man*, *spell* e così via.

A2500ux con una versione beta di AMIX è disponibile adesso a membri qualificati del programma di supporto agli sviluppatori commerciali.

Un altro bollettino parla di una High Resolution Colour Graphics Card. Questa scheda supporterà risoluzioni fino a 1024x1024 con 8 bit-plane, fornendo 256 colori (da un palette di 16 milioni di colori) più due bit-plane di overlay che forniscono altri tre colori contemporaneamente. Un processore ad alta velocità per sistemi grafici prodotto della Texas Instruments (TMS34010) eseguirà operazioni grafiche con una velocità di fino a 6 milioni di istruzioni al secondo. Una libreria grafica standard TIGA fornirà una serie di comandi grafici ad alta velocità per il TMS34010.

C'è anche un bollettino che descrive il PVA2350 Professional Video Adapter. Questo è un frame-grabber che opera in tempo reale con una risoluozione di un sessantesimo di secondo. Può catturare il campo video pari, dispari o entrambi a colori e supporta tutte le risoluzioni grafiche di Amiga, incluso il modo HAM. La scheda funziona anche da digitalizzatore e da genlock con entrambe le uscite RGB e video composito.

Infine, la Commodore ha distribuito un bollettino riguardante l'A590 Hard Disk controller con espansione di memoria. L'A590 è un hard-drive da 20 Mbyte esplicitamente progettato per A500. È autoconfigurante e auto-boot con la versione 1.3 del sistema operativo di Amiga. Opera come unità DMA, si collega alla porta di espansione laterale di A500 e possiede un connettore SCSI stile Mac sul retro, per permettere il collegamento di drive aggiuntivi. L'A590 contiene gli zoccoli per allogiare fino a 2 Mega di RAM d'espansione. Sebbene non sia scritto nel bollettino, mi sembra di aver capito che l'A590 funzionerà anche con A1000.

Alcune delle apparecchiature di cui abbiamo parlato più sopra sono state annunciate ufficialmente come prodotti disponibili in breve tempo per il mercato europeo durante l'ultima mostra CeBIT che si è tenuta a Hanover, West Germany. Questo significa che dovrebbero essere disponibili in un tempo breve anche negli US.

Anche se ho brontolato su alcune sue politiche di assistenza, non posso che dire bene del dipartimento di ingegneria della Commodore. I bollettini che abbiamo visto mi dicono che la Commodore sta lavorando sodo per fare di Amiga un serio contendente nel mercato dei computer. Questo non può che significare buone cose per quelli di noi che posseggono già un Amiga.

Attenzione!

Per motivi indipendenti dalla nostra volontà, non è possibile accludere i listati dell'articolo "Algoritmo di riempimento veloce", a pag. 27 di questo stesso numero, nel dischetto AmiTrans 6.

Ce ne scusiamo con i lettori.

Lo standard IFF

di Leonardo Fei

Non si può usare un programma Amiga (a eccezione, forse, dei giochi) senza imbattersi prima o poi nella parola magica IFF. In effetti, anche senza sapere esattamente quale ne siano le origini o il significato, tutti abbiamo potuto verificare empiricamente che IFF vuole dire compatibilità (ci sono ovviamente le solite eccezioni che confermano la regola). La sigla IFF è stata coniata usando le iniziali di *Interchange File Format* (formato per l'interscambio di file) quando Amiga era ancora poco più di un prototipo. La lungimiranza (una volta tanto!) della Commodore e della maggiore software house americana legata per tradizione ai computer Commodore, la EA (Electronic Arts), ha riunito a tavolino le menti dei migliori programmatori per formulare uno standard che permettesse di scambiare i dati più diversi fra un programma e l'altro, fornendo un supporto potente e flessibile che potesse essere usato e ampliato nel futuro senza dovere rinunciare a niente di ciò che fosse già in uso. Gli effetti più spettacolari (è proprio il caso di dirlo) si sono avuti nel campo delle applicazioni grafiche, seguiti a ruota dai programmi musicali. A coloro che sono approdati ad Amiga, soprattutto provenendo dal mondo degli IBM compatibili, non potrà essere passata inosservata la facilità con la quale si crea un'immagine con un programma e la si manipola con uno o più programmi diversi, ognuno di questi specializzato in una serie di operazioni ed effetti particolari.

Grazie all'adozione dello standard IFF, i programmi grafici possono scambiarsi immagini o loro parti, informazioni sulla loro posizione, sui colori utilizzati, sugli eventuali cicli di colore da usare per *animare* i disegni e altre informazioni, nella rassicurante consapevolezza che queste verranno usate senza essere confuse o malinterpretate da altri programmi, creando un ambiente di lavoro unico in quanto a flessibilità e affidabilità. Chi si sia in precedenza dibattuto fra ridde di formati, uno diverso dall'altro, (ci vengono subito in mente CGA, EGA, VGA, ecc.) che richiedono appositi programmi di conversione e funzioni di "import-export" per essere passati da un programma all'altro, non può non rendersi conto di quanto lo standard IFF ci abbia semplificato la vita. L'esempio più comune consiste, appunto, nel creare o digitalizzare un'immagine con uno degli appositi programmi, passarla poi a uno o più programmi specializzati per le elaborazioni, trucchi ed effetti speciali e magari utilizzare l'immagine risultante in un programma di animazione. Alcuni dei disegni usati per le copertine di questa rivista sono passati anche attraverso tre o quattro programmi prima che il risultato fosse giudicato soddisfacente. Nel campo musicale lo standard IFF ha permesso che strumenti creati o campionati con un programma venissero utilizzati da un programma diverso, magari più versatile rispetto all'altro nella scrittura di uno spartito musicale.

Ma vediamo adesso in cosa consiste esattamente lo standard IFF. Innanzitutto vengono definiti due livelli. Il primo consiste nel contenitore, nel guscio, che è comune a tutti i file IFF. Questo livello descrive, in pratica, che aspetto deve avere un file IFF e come deve essere interpretato. Fanno parte del primo livello la definizione degli identificatori FORM, LIST, 'CAT ' e PROP. Il secondo livello tratta, caso per caso, i tipi specifici di file IFF, quali ILBM (per le immagini), SMUS (per la musica), FTXT (per i testi), 8SVX (per la sintesi vocale campionata a 8 bit) e così via. Un file IFF è costituito da blocchi di lunghezza variabile chiamati 'chunk' (letteralmente: pezzo) che racchiudono al loro interno i dati. Un chunk è costituito da un identificatore di 4 byte che ne individua il tipo, da una longword (ancora 4 byte) che indica la lunghezza della sezione dati e infine dalla sezione dati stessa.

Prendiamo, per esempio, un chunk chiamato BMHD; la sua struttura si presenta come:

```
offset  contenuto                 significato
0:      'BMHD'                    identificatore
4:      20                        lunghezza dei dati
8:      320, 256, 0, 0, ....      dati (20 byte)
27:     ultimo byte del chunk
28:     eventuale chunk successivo
```

La colonna a sinistra riporta l'offset, la distanza, dall'inizio del chunk. Dalla posizione 0 alla posizione 3 troviamo i quattro caratteri 'B', 'M', 'H', 'D', che identificano il tipo di chunk. Si possono usare a questo scopo i caratteri ASCII aventi codice compreso fra il carattere spazio " " (32, 0x20) e il carattere tilde "~" (126, 0x7E). Il nome di un identificatore non può cominciare con uno spazio e bisogna rispettare le maiuscole/minuscole. La lunghezza di questo identificatore risulta particolarmente conveniente in quanto quattro byte formano una longword (32 bit) che può essere riconosciuta con una sola istruzione del microprocessore. La longword successiva, dalla posizione 4 alla 7, contiene il contatore dei byte che compongono la parte dati e quindi ci dice indirettamente dove finisce il chunk, ovvero, dove saltare per trovare il chunk successivo (qualora ci sia). Nel nostro esempio il contatore ci dice che sono presenti venti byte di dati, che si troveranno quindi dalla posizione 8 alla posizione 27 compresa. Un eventuale chunk successivo inizierebbe alla posizione 28. Nel caso di chunk di lunghezza dispari, per esempio lo stesso visto prima ma con 19 byte di dati, la sezione dati deve essere seguita da un byte di riempimento (pad) contenente zero, in modo che il file sia sempre di lunghezza pari e che l'eventuale chunk successivo inizi a un indirizzo pari. Questo è importante perché il microprocessore può gestire valori di 16 e 32 bit solo se iniziano su un indirizzo pari. In tal caso il chunk avrebbe questo aspetto:

```
offset   contenuto                 significato
0:       'BMHD'                    identificatore
4:       19                        lunghezza dei dati
8:       320, 256, 0, 0, ....      dati (19 byte)
26:      utimo byte del chunk
27:      0                         byte di riempimento
28:      eventuale chunk successivo
```

A questo punto, per costruire un file IFF è sufficiente prendere un chunk 'contenitore' che funge da guscio e che viene chiamato FORM, inserire all'interno di questo guscio un secondo identificatore che indica il tipo del file IFF e farlo seguire da tutti i chunk che vogliamo, uno dopo l'altro. Le dimensioni della parte dati di ogni chunk ci dicono quanti byte saltare per arrivare al chunk successivo. In sostanza il chunk FORM non è altro che un normale chunk i cui primi 4 byte della zona dati costituiscono l'identificatore del tipo di file e i cui rimanenti dati sono costituiti da altri chunk. Vediamo un esempio pratico costituito da un file IFF di tipo ILBM che può essere letto da un qualsiasi programma grafico:

```
offset   contenuto                 significato
0:       'FORM'                    identificatore
4:       10294                     lunghezza dei dati
8:       'ILBM'                    tipo di file
12:      'BMHD'                    identificatore
16:      20                        lunghezza dei dati
20:      320, 256, 0, 0, 1, .... dati (20 byte)
40:      'CMAP'                    identificatore
44:      6                         lunghezza dei dati
48:      0, 0, 0, 240, 240, 240    dati (6 byte)
54:      'BODY'                    identificatore
58:      10240                     lunghezza dei dati
62:      0, 0, 0, ....             dati (10240 byte)
10301:   ultimo byte del chunk BODY e del chunk FORM
```

Schematicamente la sua struttura può essere così rappresentata:

```
FORM ILBM {
  BMHD
  CMAP
  BODY
}
```

Questo file IFF di tipo ILBM contiene tre chunk che descrivono completamente un'immagine. Il primo chunk, BMHD, contiene le informazioni relative alle dimensioni dell'immagine, la sua posizione all'interno dello schermo, il numero di bitplane ecc. Il secondo, CMAP, contiene le informazioni relative alla mappa dei colori usati in questa immagine. Ciascun colore è codificato da tre byte, R, G e B. Siccome nel nostro esempio abbiamo un solo bitplane, i colori sono soltanto due, il nero (0, 0, 0) e il bianco (240, 240, 240). Il terzo e ultimo chunk, BODY, contiene i dati del bitplane vero e proprio.

Tutti i file IFF iniziano con l'identificatore FORM, che contraddistingue il chunk 'contenitore', seguito dalla lunghezza del file (meno gli otto byte iniziali). Subito dopo, il secondo identificatore specifica il tipo di file IFF (riferendosi a un file IFF di un certo tipo, per esempio ILBM, possiamo dire indifferentemente file IFF ILBM o FORM ILBM). All'interno della sezione dati del chunk FORM troviamo tutti gli altri chunk *normali* che vengono usati per scambiare le informazioni. Le uniche eccezioni a questa regola sono i casi in cui un file IFF inizia con l'identificatore 'CAT ' o LIST. Tralasciamo, per il momento, di approfondirne il significato.

In definitiva, possiamo considerare i chunk *normali* BMHD, CMAP e BODY come dei mattoncini, ognuno dei quali porta una parte di informazione dell'insieme. Questi mattoncini vengono poi inseriti in un *guscio* (il chunk FORM) che rappresenta il veicolo grazie al quale potranno essere riconosciuti e quindi scambiati fra un programma e l'altro.

Abbiamo visto prima la struttura del FORM di tipo ILBM contenente i chunk BMHD, CMAP e BODY. Bisogna tenere separati i concetti di FORM e di chunk, in quanto ogni chunk viene *registrato* per un determinato tipo di FORM e ha senso solo in quel preciso FORM. Il fatto che un chunk sia registrato significa che il suo nome ed eventualmente la descrizione del suo contenuto sono stati depositati presso la Commodore, che provvede a tenere un elenco aggiornato disponibile a chiunque ne faccia richiesta. Il processo di registrazione dei chunk è controllato dalla Commodore alla quale vanno mandate le proposte per la creazione di nuovi tipi di FORM e chunk. Questo permette di evitare che due programmatori assegnino, l'uno all'insaputa dell'altro, lo stesso nome a nuovi chunk o FORM. L'autore può decidere se rendere pubblica o meno la descrizione dettagliata del contenuto del suo chunk o del suo file FORM. In generale è nel comune interesse che queste informazioni vengano rese di pubblico dominio in modo che altri possano interpretare i nuovi chunk ed eventualmente utilizzarli nei propri programmi, cosa che aiuta sicuramente anche la diffusione del programma originale a beneficio del suo autore. Rimane comunque l'obbligo di registrare almeno il nome del nuovo chunk, pena possibili conflitti di interpretazione fra il proprio chunk "non dichiarato" e quello, dichiarato o meno, di qualcun'altro.

I tre chunk che abbiamo visto sono solo alcuni di quelli registrati per file IFF di tipo ILBM. Questo vuole dire che possono apparire solo all'interno di un FORM ILBM. In teoria possiamo registrare un nuovo chunk di tipo CMAP che compaia nei FORM di tipo SMUS e assegnarli un significato completamente diverso dal già esistente chunk CMAP che appare nei FORM di tipo ILBM. Nella pratica, per evitare inutili confusioni, si evita di riutilizzare il nome di un chunk già esistente per contraddistinguerne un altro dal significato diverso all'interno di un altro FORM. Può essere utile, invece, registrare lo stesso chunk all'interno di diversi tipi di FORM, così come succede per i chunk AUTH, '(c) ', NAME, ANNO e altri, registrati per FORM di tipo SMUS e di tipo 8SVX. Questi chunk sono stati definiti per potere includere rispettivamente il nome dell'autore, le note di copyright, il titolo di un'opera e le eventuali annotazioni in un file IFF contenente uno spartito musicale o suoni campionati. Nella pratica, alcuni programmatori hanno esteso l'uso di questi chunk anche all'interno di FORM di tipo ILBM e altri, in quanto un disegno, un'immagine campionata o quant'altro possa essere considerato proprietà artistica di qualcuno deve essere, giustamente, identificato e protetto.

Quando lo standard IFF è stato creato nel 1985, sono stati standardizzati i quattro tipi di FORM a cui abbiamo già accennato (ILBM, SMUS, FTXT, 8SVX), ognuno con la sua serie di chunk. Da allora sono stati registrati in forma pubblica (con descrizione del formato accessibile a tutti) svariati nuovi tipi di FORM per rispondere alle esigenze più diverse. Ne citiamo solo alcuni per dare un'idea dell'attività che ferve attorno allo standard IFF:

```
ACBM - immagini in formato AmigaBasic
AIFF - audio IFF in formato Apple
ANBM - animazioni, Deluxe Video
ANIM - animazioni, Videoscape-3D, DPaintIII
BANK - musica MIDI, SoundQuest
HEAD - Flow, New Horizons Software
MIDI - musica, Circum Design
PGTB - program traceback, Lattice
SYTH - musica MIDI, SoundQuest
WORD - testo, Prowrite
```

Qualcuno ha anche registrato nuovi tipi di FORM mantenendo però private le informazioni sul loro contenuto:

```
C100 - testo, Cloanto Italia
PDEF - grafica, Deluxe Print
SHAK - testo, Shakespeare
VDEO - animazione, Deluxe Video
```

I programmatori e le software house lavorano continuamente su nuovi tipi di FORM e di chunk che sono quindi in attesa di essere finiti e resi pubblici. Esiste, infatti, una lista piuttosto lunga di nomi temporaneamente riservati a questo scopo.

Sebbene la parte più appariscente dell'attività stia nella creazione di nuovi tipi di FORM che soddisfino le esigenze di applicazioni diverse, bisogna tenere presente che sono stati anche creati nuovi chunk per i FORM già esistenti, per soddisfare il bisogno di ampliamento e miglioramento all'interno di applicazioni già contemplate dallo standard IFF. Prendendo come esempio il FORM di tipo ILBM, quando è stato definito conteneva i seguenti chunk:

```
BMHD - informazioni sui bitmap
CMAP - mappa dei colori
GRAB - punto caldo
DEST - Planepick
SPRT - informazioni sugli sprite
CAMG - modi di visualizzazione
CCRT - ciclo di colori (DPaint)
CRNG - ciclo di colori (Graphicraft)
BODY - dati dei bitplane
```

Il sorgere di nuove idee e di nuove esigenze ha determinato l'espansione del formato ILBM con questi nuovi chunk:

```
DPPV - prospettiva (DPaintII)
DGVW - DigiView
BHSM - Photon Paint
BHCP - Photon Paint
BHBA - Photon Paint
```

L'aggiunta di nuovi chunk a un FORM già in uso è possibile grazie alla modularità dello standard. Quando un programma legge un file IFF, ne esamina il contenuto alla ricerca dei chunk che gli servono e ignora quelli superflui o quelli che non riesce a identificare. Usa cioè quello che gli interessa. In questo modo, lo standard IFF, oltre a facilitare lo scambio di dati fra programmi diversi, rende più semplice il mantenimento della compatibilità fra le versioni più aggiornate di uno stesso programma. Il programma Deluxe Paint II, ad esempio, può creare un FORM ILBM all'interno del quale si trovano i chunk BMHD, CMAP, DPPV, CRNG, CAMG e BODY. Il file così creato può essere letto da un qualsiasi altro programma, che per la visualizzazione pura e semplice dell'immagine, senza cicli di colore, ha bisogno solamente dei chunk BMHD, CMAP, BODY. Gli altri possono venire ignorati. Grazie alla tecnica modulare con cui è composto ciascuno dei suoi file, lo standard IFF si adatta bene a qualunque tipo di informazione. Questo appare ancora più evidente soprattutto se si considera la possibilità di nidificare i FORM uno dentro l'altro (come succede salvando un'animazione da Deluxe Paint III, dove FORM ANIM contiene FORM ILBM) e se si prendono in considerazione gli altri chunk 'contenitori' 'CAT ', LIST e PROP. 'CAT ' permette di concatenare una serie di FORM. Per ottenere uno slide-show si potrebbe infatti costruire un 'CAT ' con tanti FORM ILBM, uno dietro l'altro, al suo interno. LIST è simile a 'CAT ', ma si usa insieme al chunk 'contenitore' PROP che appare al suo interno. Se disegnamo una serie di immagini usando sempre lo stesso palette di colori e la stessa risoluzione, è inutile salvare una serie di FORM ILBM che contengano ognuno gli stessi chunk BMHD, CMAP e CAMG. PROP permette di evitare queste ripetizioni, dichiarando all'interno di un LIST una serie di chunk le cui proprietà valgono per tutti i FORM contenuti nel LIST stesso. Nel nostro caso potremmo quindi mettere dentro PROP i chunk BMHD, CMAP e CAMG. Tutti i FORM ILBM che seguono PROP potranno limitarsi ad avere solo il chunk BODY al loro interno. Schematicamente:

```
LIST ILBM {
  PROP ILBM {
    BMHD
    CMAP
    CAMG
  }
  FORM ILBM {
    BODY
  }
  FORM ILBM {
    BODY
  }
.........
.........
}
```

Esaminiamo adesso il listato del programma *IffExam* che si trova alla fine di questo articolo. Il programma è una versione modificata del programma *IffDump* di Matt Dillon. Il suo scopo è quello di esaminare un file, che sia sicuramente conforme allo standard IFF, stampando l'elenco dei chunk e la loro lunghezza. Utilizzando il flag *-v* verranno stampati in forma esadecimale anche i dati contenuti in ogni chunk. La sintassi d'uso

prevede che il programma sia invocato con il nome del file da esaminare, eventualmente seguito dal flag -v. Il listato inizia con la solita parte che si occupa di leggere e interpretare gli argomenti presenti sulla linea di comando. Il file IFF viene quindi aperto e ne viene ricavata la lunghezza, poi si chiama la routine *iffdecode* che rappresenta il cuore centrale del programma. In questa routine viene letto l'inizio di un chunk per stabilire se si tratta di un chunk 'contenitore' (del tipo FORM, LIST, 'CAT ' o PROP) oppure se si tratta di un chunk semplice. Nel primo caso viene letto il secondo identificatore che indica il tipo di file e quindi la funzione iffdecode invoca ricorsivamente sè stessa per esaminare il chunk nidificato all'interno del precedente. Se anche il secondo chunk è di tipo 'contenitore', il processo si ripete con un altra invocazione di iffdecode e così via, fino a quando non si raggiunge l'ultimo livello del nido. A questo punto la variabile *ok* viene messa a 1 e si procede a leggere i chunk 'normali' all'interno dell'ultimo livello, stampandone eventualmente il contenuto se la variabile *verbose* risulta uguale a TRUE. Ogni volta che si esaurisce il contenuto del livello attuale, l'ultima chiamata a iffdecode restituisce il controllo a quella precedente, che continua l'esame del livello superiore. Durante l'esecuzione della funzione *iffdecode* vengono chiamate le altre funzioni di appoggio che si occupano dei 'dettagli'. La funzione *hexdump* viene invocata per stampare in forma esadecimale la sezione dati di un chunk. Gli argomenti che vengono forniti specificano la posizione attuale all'interno del file IFF sotto esame, la lunghezza della parte dati da stampare e la lunghezza della stringa di spazi che serve a indentare l'output verso destra. Quest'ultima viene incrementata ogni volta che si scende di un livello nella nidificazione dei chunk e viene diminuita quando si risale, in modo da rendere l'operazione graficamente chiara. I dati vengono stampati quattro byte alla volta per agevolarne la lettura. Se la lunghezza della parte dati non è esattamente un multiplo di quattro, le istruzioni successive si occupano di stampare i dati rimanenti e l'eventuale byte di riempimento (pad), che secondo lo standard IFF dovrebbe essere sempre a zero. La funzione *tabstr* riceve come argomento il numero degli spazi necessari per l'indentazione della stampa e restituisce la stringa appositamente creata. La funzione *chkread* si occupa di leggere la successiva porzione del file sotto esame, preoccupandosi di segnalare un'eventuale interruzione inaspettata del file.

Per provare il programma IffExam, abbiamo salvato un'animazione di due fotogrammi con Deluxe Paint III. Ecco il risultato (senza flag -v):

```
chunk FORM (1216) tipo ANIM
chunk FORM (808) tipo ILBM
chunk BMHD (20)
chunk CMAP (6)
chunk DPPS (110)
chunk CRNG (8)
chunk CRNG (8)
chunk CRNG (8)
chunk CRNG (8)
chunk CRNG (8)
chunk CRNG (8)
chunk DPAN (8)
chunk CAMG (4)
chunk BODY (512)
chunk FORM (124) tipo ILBM
chunk ANHD (40)
chunk DLTA (64)
chunk FORM (124) tipo ILBM
chunk ANHD (40)
chunk DLTA (64)
chunk FORM (124) tipo ILBM
chunk ANHD (40)
chunk DLTA (64)
```

Le informazioni relative allo stato dello standard IFF, le descrizioni dei chunk registrati pubblicamente, informazioni varie e molti esempi accompagnati dal relativo codice sorgente si possono trovare sul disco 'IFF' che viene periodicamente aggiornato e distribuito dalla Commodore come materiale di pubblico dominio. Questo disco contiene infatti un settore 'documenti' nel quale sono elencati tutti i chunk e le loro eventuali descrizioni, oltre a note, suggerimenti ed esempi illuminanti. Un settore 'eseguibili' contiene alcuni programmi di utilità, da IFFCheck, che controlla la validità sintattica di un file presunto IFF, a ZapIcon che trasforma un brush creato da DPaint in un'icona. I sorgenti di questi programmi sono contenuti in un settore 'sorgenti', insieme a molti altri pezzi di codice sorgente e routine varie che costituiscono i mattoni da inglobare a piacimento nel proprio programma per leggere e scrivere file IFF di qualsiasi tipo.

In passato Fred Fish ha distribuito le prime due versioni del disco 'IFF' preparato dalla Commodore (la più recente delle due è contenuta nel Fish #64). In occasione del DEVCON del novembre 88 è stato distribuito agli sviluppatori l'ultima e più aggiornata versione del disco 'IFF', il cui materiale appare anche, quasi interamente, nel manuale pubblicato nel gennaio 89 dalla Addison-Wesley con il titolo "Amiga ROM Kernel Reference Manual: Includes & Autodocs" ISBN 0-201-18177-0. In alternativa, la documentazione e il disco 'IFF' possono essere ordinati, tramite assegno di 25 dollari intestato a 'Commodore Business Machines', richiedendo "IFF Docs and Disk" a:

CATS - Orders
Brenda Billings
CBM 1200 Wilson Drive
West Chester, PA. 19380 U.S.A.

```
/* IffExam - tratto da IffDump di Matt Dillon,
 *           modificato da Leonardo Fei.
 * NOTA:  Questo programma non controlla che il
 *        formato del file IFF sia valido (per
 *        esempio chunk PROP solo all'interno di
 *        chunk LIST, ecc.). Il suo scopo e' quello
 *        di visualizzare file IFF validi.
 * Compilatore: Manx 3.6a
 *              cc +p IffExam.c
 *              ln IffExam.o -lcl32
 */

#include <exec/types.h>
```

```
#include <stdio.h>

#define MAKEID(a,b,c,d) ((a)<<24|(b)<<16|(c)<<8|(d))

#define IFF_FORM     MAKEID('F','O','R','M')
#define IFF_LIST     MAKEID('L','I','S','T')
#define IFF_PROP     MAKEID('P','R','O','P')
#define IFF_CAT      MAKEID('C','A','T',' ')
#define IFF_FILLER   MAKEID(' ',' ',' ',' ')


char *tabstr();

void chkread();
void hexdump();

BOOL verbose = FALSE;
FILE *file;

main(argc, argv)
   int argc;
   char *argv[];
{
char *string;
long flength;

      /* controlla gli argomenti */
   if (argc < 2)
      {
      fprintf(stderr, "\nUso: %s <file> [-v]\n",
             argv[0]);
      exit (20);
      }

   if (argc == 3)
      {
      string = argv[2];
      if (*string++ == '-')
         {
         switch (*string)
            {
            case 'v':
               verbose = TRUE;
               break;

            default:
               fprintf(stderr,
                      "'%c' opzione non valida\n",
                      *string);

            }
         }
      else
         {
         fprintf(stderr, "\nUso: %s <file> [-v]\n",
                argv[0]);
         exit (20);
         }
      }

      /* apre il file da esaminare */
   if ((file = fopen(argv[1], "r")) == 0)
      {
      fprintf(stderr,
             "\nNon posso aprire il file %s!\n",
             argv[1]);
      exit (20);
      }

   fseek(file, 0, 2);
   flength = ftell(file);   /* lunghezza del file */
   fseek(file, 0, 0);

   printf("\n");

   iffdecode(0, flength);   /* esamina il file */

   fclose(file);

}

/*************************************************/

iffdecode(tab, flength)
   int tab;
   long flength;
{
long chunk[2];
long curpos, bytes, curend;
long data;
char ok;
char count = 0;

   for (;;)
      {
      curpos = ftell(file);  /* posizione attuale */
      if (curpos == flength)
         return (1);

      if (count++ && tab == 0)
         {
         puts ("Attezione: caratteri spurii dopo la
fine del file IFF\n");
         return (0);
         }

      if (curpos + 8 > flength)
         {
         puts ("Errore: fine prematura del file
all'interno di un header\n");
         return (0);
         }

      chkread(chunk, 8);  /* ID e lunghezza chunk */
      curpos += 8;
      bytes = chunk[1];  /* lunghezza chunk */
      curend = curpos + bytes;
      printf ("%schunk %4.4s (%u)", tabstr(tab),
             &chunk[0], bytes);
```

```
      if (curend > flength)
         {
         puts ("\nErrore: fine prematura del file
all'interno di un header\n");
         return (0);
         }

      ok = 0;
      switch(chunk[0])
         {
         case IFF_FORM:
         case IFF_LIST:
         case IFF_CAT:
         case IFF_PROP:
            chkread(&data, 4);
            curpos += 4;
            bytes  -= 4;
            printf(" tipo %4.4s\n", &data);
            iffdecode(tab + 4, curend);
            break;

         case IFF_FILLER:
            puts (" (chunk di riempimento)");
            if (verbose)
               hexdump(curpos, bytes, tab);
            break;

         default:
            ok = 1;
            break;

         }

      if (ok)
         {
         if (verbose)
            hexdump(curpos, bytes, tab);
         }

      if (curend & 1)
         {
         ++curend;
         if (!verbose)
            printf("\n%s(byte di riempimento)",
                    tabstr(tab));
         }

      printf("\n");

      fseek(file, curend, 0);
      }

} /* end of iffdecode */

/***********************************************/

  void
hexdump(curpos, length, tab)
   long curpos, length;
   int tab;
{
long count, done=0;
UBYTE i = 0;
ULONG data;

   fseek(file, curpos, 0);

   if (fseek(file, length, 1))
      {
      printf("Errore nella lettura del file!\n");
      exit (20);
      }
   else
      fseek(file, -length, 1);

   printf("\n%s  %04.4x: ", tabstr(tab), done);

      /* legge e stampa 4 byte alla volta */
   count = length / 4;
   while (count--)
      {
      chkread(&data, 4);

      printf("%08.8x ", data);
      done += 4;

      if (++i == 6)
         {
         printf("\n%s  %04.4x: ",tabstr(tab),done);
         i = 0;
         }
      }

      /* legge e stampa eventuali byte rimanenti */
   count = length % 4;
   if (count != 0)
      {
      while (count--)
         {
         chkread(&i, 1);

         printf("%02.2x", i);
         done++;
         }
      }
/* legge e stampa eventuale byte di pad */
   count = length % 2;
   if (count != 0)
      {
      chkread(&i, 1);

      printf(
      "\n%s  %04.4x: %02.2x (byte di riempimento)",
      tabstr(tab), done, i);
      }

   printf("\n");
} /* end of hexdump */
```

Continua a pagina 82

Algoritmo di riempimento veloce

Velocizzate il riempimento di aree fino a cinque volte!

di Danny Ross

Danny Ross sta frequentando l'ultimo anno di Scienze dell'Informazione. Il suo tempo libero si divide fra lo sviluppo di algoritmi di incanalamento per network di transputer, fra la programmazione di Amiga come freelance e fra la scrittura di software per il PC. Vivendo in una casetta in stile Gotico nel Galles del sud con altri due maniaci della programmazione, la sua idea di rilassamento consiste nell'esplorare tutte le ultime chiamate delle librerie di Amiga!

Fino a poco tempo fa, stavo lavorando alla conversione di un piccolo programma per PC, scritto da un collega in TURBO C, che veniva usato per generare paesaggi frattali partendo da modelli di profilo. In quella fase di sviluppo il programma utilizzava un sistema controllato da mouse che permetteva all'utente di disegnare il profilo della mappa, dalla quale sarebbe stato generato il paesaggio. La conversione stava filando via abbastanza liscia, fino a quando non raggiunsi la parte dove il mio collega aveva utilizzato la routine di riempimento di colore del TURBO C. In questo punto la versione Amiga sembrava semplicemente fermarsi bruscamente.

Il problema consisteva nel fatto che la routine *Flood()* della graphics.library non riusciva a stare all'altezza della velocità dell'equivalente routine contenuta nel TURBO C e le cose cominciavano proprio ad assumere un brutto aspetto. Essendo un fervido devoto della graphics.library mi risultava molto difficile accettare che fosse così lenta, ma alcuni esperimenti dimostrarono presto che la routine *Flood()* era realmente così letargica com'era apparso all'inizio e quindi iniziai a cercare un'alternativa.

Il presente articolo è composto della descrizione della funzione *Flood()* incorporata in Amiga (con alcuni tentativi di spiegare le sue scarse prestazioni), della descrizione di un algoritmo sostitutivo per effettuare le stesse operazioni e, infine, di un'implementazione scritta in C e assembly che supera le prestazioni della funzione *Flood* di un fattore che si aggira intorno al 500%.

Flood()

Facendo parte della *graphics.library*, Flood() esegue il compito, raramente richiesto, di riempire una forma sconosciuta con un determinato colore. Sebbene venga spesso lasciata inutilizzata a favore della routine *AreaFill*, *Flood()*, quando viene richiesta, è molto utile. Necessita solo delle coordinate (X,Y) di un punto all'interno della forma da colorare.

AreaFill, comunque, è in grado di utilizzare il Blitter per svolgere il suo lavoro di riempimento, mentre *Flood* usa molto la CPU. Il risultato è, sfortunatamente, che *AreaFill* è estremamente veloce mentre *Flood()* è diabolicamente lenta.

Benché l'algoritmo usato da *Flood()* sia buono, l'implementazione soffre di alcuni problemi. Se siete mai stati spinti a usare *ReadPixel()* o *WritePixel()* potreste avere notato che sono terribilmente lente. La ragione di tutto questo sta nel fatto che la complessità delle strutture grafiche di Amiga introduce un tempo supplementare (overhead) molto alto, per ogni singola operazione di lettura o scrittura. Entrambe le funzioni *ReadPixel* e *WritePixel* sono soggette a questo tempo supplementare e sembra che *Flood()*, usando queste funzioni, ne soffra altrettanto.

L'algoritmo di vicinanza di linea

Se dobbiamo scrivere una routine sostitutiva di *Flood()* avremo bisogno di un algoritmo per il riempimento di aree che svolga questo compito con delle prestazioni ragionevoli. Questo algoritmo si chiama *line adjacency* (vicinanza di linea).

La strategia di base del riempimento operato da questo algoritmo sta nel localizzare ogni gruppo di pixel connessi orizzontalmente all'interno della regione. L'algoritmo parte da un pixel d'origine che è contenuto all'interno della regione ed effettua una scansione verso sinistra e verso destra per trovare l'inizio e la fine della riga sulla quale si trova il pixel in questione. L'intera riga viene quindi riempita con il nuovo colore.

L'algoritmo procede localizzando tutti i gruppi di pixel connessi orizzontalmente che risultano verticalmente adiacenti al gruppo che è stato appena esaminato. Ogni volta che *LineAdjacencyFill()* trova un gruppo di pixel adiacenti che non sono ancora stati riempiti, chiama ricorsivamente sè stessa. L'algoritmo termina quando tutti i pixel interni sono stati riempiti.

La figura 1 mostra il percorso effettuato da *LineAdjacencyFill()*. La routine viene chiamata inizialmente con le coordinate del pixel d'origine in E7 e con la direzione iniziale verso il basso. L'algoritmo effettua una scansione verso sinistra e verso destra per trovare l'inizio e la fine del primo gruppo, il quale viene poi riempito usando la funzione *Draw()* della graphics.library, molto più veloce della *WritePixel*.

Il prossimo passo dell'algoritmo consiste nel trovare e riempire segmenti di linea adiacenti nella stessa direzione. Siccome il primo gruppo è stato E6- E9, *LineAdjacencyFill()* si prepara a verificare tutti i pixel nel segmento F6-F9. In F6 viene controllato se il pixel è adatto a essere riempito (e se fa quindi parte di un gruppo orizzontale). Se la risposta è positiva, *LineAdjacencyFill()* viene chiamata ricorsivamente per riempirlo.

Al suo ritorno dalla chiamata, l'algoritmo continua lungo la linea F6-F9; la precedente invocazione di *LineAdjacencyFill()*, comunque, ha effettuato una scansione sinistra-destra sulla riga F da F6, così, come bonus, possiamo saltare oltre il limite destro nella consapevolezza che i pixel saltati sono già stati esaminati. Nell'esempio, la seconda chiamata avrà trovato un segmento di linea F2-F7 e l'avrà riempito, quindi possiamo ricominciare da F8.

L'algoritmo è completamente ricorsivo e quindi la seconda invocazione continuerà dopo avere trovato F2-F8 e comincerà a controllare i pixel fra G2- G7. Questo genererà una terza chiamata (perché G2 è un candidato accettabile) e il segmento G2-G11 verrà riempito. Ancora una volta, H2-H11 verrà controllato ma nessuno dei pixel provocherà la chiamata di *LineAdjacencyFill()*, in quanto tutti i pixel in questione formano il confine della regione. La terza invocazione, allora, passa alla fase successiva e comincia a controllare nella direzione opposta.

Per ottenere il massimo della velocità, il controllo in direzione opposta viene effettuato solo su quegli intervalli che sono al di fuori del segmento genitore (le variabili del programma chiamate *prevxl* e *prevxr*). Nell'esempio, il pixel attualmente all'estrema sinistra è G2, il quale non si trova ulteriormente a sinistra di F2 della linea precedente, quindi nessuna azione viene intrapresa.

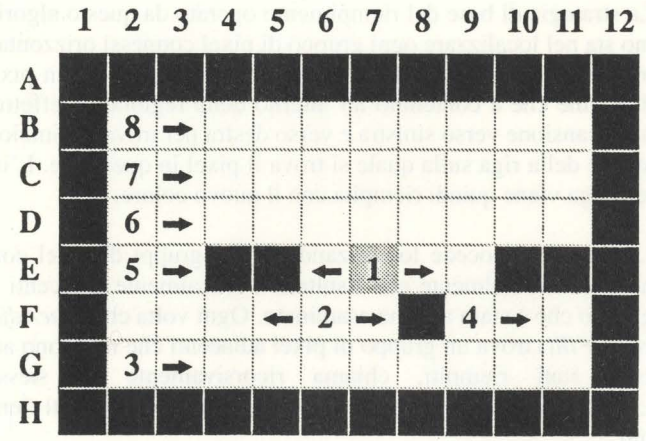


Figura 1

Sull'estremità destra, viene effettuato un controllo fra F8-F11, risultante nella chiamata di *LineAdjacencyFill()* in F9, per riempire il segmento F9-F11. Questa chiamata ha una vita molto corta: il primo controllo nella stessa direzione non trova candidati accettabili (E9 è già stato riempito) e il controllo in direzione opposta non genera risultati degni di nota. Quest'ultima chiamata termina, quindi, permettendo alla terza invocazione di continuare la sua ricerca in direzione opposta, ricominciando da F12. Questa locazione si trova però fuori portata, quindi anche la terza invocazione termina.

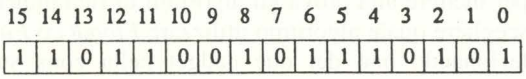
Adesso continua la seconda invocazione (che fatica!), abbandonando la ricerca nella stessa direzione (doveva solo controllare G2-G7, ma la terza invocazione ha già trovato e riempito G2-G11) e comincia il controllo in direzione opposta. Questa azione porta all'individuazione di E2 e fa partire una serie di chiamate a *LineAdjacencyFill()* per riempire E2-E3, D2-D11, C2-C11 e B2-B11.

Alla fine ritornano tutte di nuovo alla seconda invocazione che, saltando un controllo senza scopo in direzione invertita sulla destra, termina e restituisce il controllo alla prima invocazione. Quest'ultima incomincia il suo controllo in direzione opposta da D6 e D8-D9 (la prima invocazione fa sì che le variabili *prevxl* e prevxr siano uguali alla coordinata X d'origine, E7 nell'esempio). Quando questo succede, l'intera riga D risulta già riempita e quindi anche la prima invocazione termina. L'area è stata completamente attraversata.

I vantaggi di *LineAdjacencyFill()* rispetto ad alcuni degli algoritmi di riempimento più semplici, sono significativi. Il più immediato (in un'implementazione su Amiga) è che la lettura di pixel viene mantenuta al minimo indispensabile e il riempimento viene effettuato una linea alla volta. Un'implementazione C di *LineAdjacencyFill()* segue la stessa strategia di riempimento utilizzata dalla routine *Flood()* della graphics.library in modo praticamente identico (in realtà è così identico che sembra quasi certo che nel cuore di *Flood()* ci sia un algoritmo di vicinanza di linea che aspetta di liberarsi).

Ma allora perché disturbarci con tutto quel codice se in effetti non si guadagna niente? Ebbene, una delle ragioni per cui *LineAdjacencyFill()* riesce a ottenere le sue ragionevoli prestazioni risiede nella ottimizzazione che è stata ottenuta usando il riempimento di una linea alla volta. Conoscendo la posizione di inizio e di fine possiamo disegnare la linea che le collega molto più velocemente che usando ripetutamente *WritePixel()*.

Ovviamente la prossima parte che si presta a essere ottimizzata è quella che riguarda *ReadPixel*. Sebbene, infatti, l'algoritmo di vicinanza di linea riduca sensibilmente il numero delle chiamate alla funzione *ReadPixel*, alcune parti di quest'algoritmo (in particolare *scanleft* e *scanright*) poggiano ancora pesantemente sulla lettura individuale di pixel. Siccome potremmo anche trovarci a effettuare svariate centinaia di operazioni di lettura per ogni riga, risulta sensato tentare di ottimizzare i metodi utilizzati per implementare *scanleft* e *scanright*.



XP=30

therefore BitPosition = (XP mod 16)
= 14

Figura 2

La funzione streamRead()

Per produrre un rimpiazzo veloce della routine *ReadPixel()* possiamo avvantaggiarci della natura di *scanleft* e *scanright*. Sapendo che avranno bisogno di letture sequenziali e unidirezionali in posizioni Y costanti, possiamo preparare un ambiente che, dopo l'inizializzazione, può effettuare letture di un flusso di pixel in maniera molto più veloce, senza nessuno dei tempi supplementari richiesti normalmente per l'accesso alle strutture. Le routine *_initStreamRead* e *_streamRead* (listato 2) sono state scritte in assembly e rappresentano l'unica ragione dell'incremento quintuplo di prestazioni di *FastFlood* rispetto a *Flood*.

La tecnica utilizzata per effettuare la *streamRead* (lettura di un flusso, di pixel) è illustrata nelle figure 2, 3a e 3b.

Quando *_initStreamRead* viene chiamata, le viene passata una coppia di coordinate (X,Y) dalla quale viene calcolato l'offset nella *RastPort* dello schermo [52-60]. Quest'offset viene utilizzato per leggere una word da ogni bitplane dello schermo, quindi viene calcolata la posizione attuale del pixel all'interno della word [77 e figura 2].

Queste word vengono poi preparate per la lettura di flusso, spostando i pixel che devono essere letti sul bordo. Nell'esempio di figura 2, l'offset del pixel è stato calcolato in 14. Se leggiamo verso sinistra avremo bisogno dei pixel 14 e 15, mentre se leggiamo verso destra avremo bisogno dei pixel da 14 a 0. Nel caso di una inizializzazione per una lettura verso sinistra (figura 3a) la funzione *_initStreamRead()* sposta i 14 pixel della word verso destra, portando i bit 14 e 15 sul bordo.

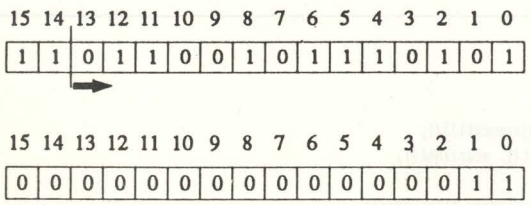


Figura 3a

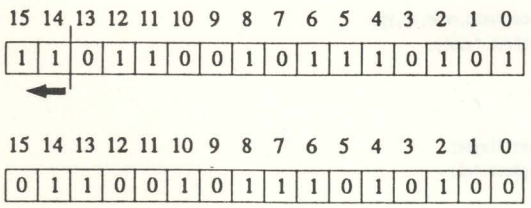


Figura 3b

Man mano che *_initStreamRead* procede, sposterà questi bit fuori dalla word in ogni bitplane (una volta per pixel) per costruire un indice dei colori. Quando i bit utili saranno stati assorbiti (due in questo caso), *_streamRead()* andrà a pescare dalla memoria la prossima word consecutiva. Seguendo lo stesso principio, una inizializzazione per una lettura verso destra (figura 3b) porterà i bit verso il bordo sinistro.

Una volta che l'inizializzazione è stata effettuata, il percorso è libero e *_streamRead()* può muoversi attraverso la linea del raster molto velocemente. Si dovrebbe prestare attenzione al fatto che, mentre l'indice dei colori viene costruito spostando i registri nel registro accumulatore [160-161 e 200-201], l'ordine nel quale si accede ai bitplane risulta di grande importanza.

All'interno della struttura BitMap si trova un array che può contenere fino a 8 puntatori, ognuno dei quali rappresenta l'indirizzo di un bitplane del display. Siccome i puntatori sono memorizzati in modo che quelli con priorità più bassa vengano prima, un traversamento normale dell'array di puntatori porterebbe alla creazione di un indice dei colori invertito e quindi sono state introdotte le linee 70-74 che servono a leggere al contrario i puntatori ai bitplane.

L'implementazione

La funzione *FastFlood()* è contenuta all'interno del file include *fastflood.h* (listato 1). Questa non è una soluzione elegante, ma rende le cose più semplici. Se utilizzate il compilatore C della Lattice e il BLink, potete rendere *FastFlood()* una funzione autonoma ed eseguirne il PRELINK insieme al file binario *fastpixel.o* prodotto dal listato 2.

Per usare *FastFlood()* dovremo prima assemblare *fastpixel.asm* per produrre il file *fastpixel.o*. A questo punto, ogni programma che richiede la funzione *FastFlood()* deve contenere la linea:

```
#include "fastflood.h"
```

all'inizio (vedi listato 3). L'intera baracca può quindi essere compilata e linkata con *fastpixel.o* e il modulo di startup standard del C.

Limitazioni

Ebbene, niente è perfetto e sfortunatamente *FastFlood()* non fa eccezione. Ci sono un paio di aspetti negativi della funzione che bisogna capire prima di utilizzarla. Per prima cosa, nella funzione *_streamRead* non viene effettuato alcun controllo sui confini della RastPort. Ho deciso di non implementarlo in quanto, nella maggior parte dei casi, risulterebbe superfluo. Avendo detto questo, se a un'operazione di riempimento viene permesso di sconfinare in un'area senza confini, allora le fun-

zioni *scanleft()* e *scanright()* andranno fuori controllo e invaderanno tutta la memoria disponibile!

Il secondo problema si manifesta quando si ha bisogno di chiamare la funzione *LockLayer()*. Siccome stiamo effettuando delle operazioni di lettura verso sinistra e verso destra, all'utente non deve essere permesso di muovere la finestra, in quanto i puntatori ai bitplane indirizzeranno ancora la stessa zona fisica dello schermo, che adesso potrebbe però essere fuori dai confini dell'area. La *LockLayer* sospenderà qualsiasi task tenti di alterare la struttura della RastPort in uso (come muovere la finestra o selezionare un gadget). Se tentate di muovere la finestra con *Flood()* noterete lo stesso effetto. *FastFlood()*, comunque, usa abbondantemente lo stack per la sua natura ricorsiva e se questo non è dimensionato a sufficienza (circa 25k dovrebbero andare bene per il programma dimostrativo) potrebbe causare un overflow. Se questo fatto dovesse accadere fra *LockLayer()* e *UnlockLayer()* potremmo ritrovarci in guai seri.

La prima limitazione è facile da superare, in quanto i confini della finestra sono conoscibili sin dall'inizio di *FastFlood()*, dove la variabili *ff_xoff* e *ff_yoff* sono inizializzate partendo dalla struttura Border all'interno della ClipRect, all'interno della Layer, all'interno della RastPort. Queste variabili possono essere passate a *_initStreamRead()* con il valore dell'attuale coordinata x mantenuto e controllato da *_streamRead()*.

La seconda limitazione è più difficile da eliminare e l'unica soluzione che ho potuto escogitare è stata quella di eseguire il Lock e l'Unlock della struttura Layer rispettivamente all'inizio e alla fine di ogni chiamata a *scanleft()* e *scanright()*. Queste funzioni non fanno chiamate ricorsive quindi sono relativamente al sicuro dalla trappola dell'overflow dello stack.

Il programma dimostrativo

Quando lanciamo il programma dimostrativo del listato 3, ci verrà presentato uno schermo scarno ma funzionale con svariati gadget e una zona per disegnare. Premendo il tasto sinistro del mouse in quest'ultima zona, potremo disegnare delle aree per mettere alla prova gli algoritmi di riempimento. Possiamo scegliere quale algoritmo utilizzare *Flood()* o *FastFlood()* selezionando uno dei due gadget nella zona superiore dello schermo.

Più in basso nello schermo si trovano altri due gadget per aumentare o diminuire il colore di riempimento e ancora più in basso due gadget per ripulire l'area di disegno e per uscire dal programma.

Il riempimento viene effettuato premendo il tasto destro del mouse in qualsiasi punto all'interno dell'area di disegno, ma fate attenzione che non ci sia un buco nel profilo della nostra zona, altrimenti il riempimento coinvolgerà tutta l'area.

Conclusione

La routine *FastFlood()*, combinata con le funzioni contenute in *fastpixel.asm*, costituisce un efficace rimpiazzo per la routine *Flood()*, fornendo prestazioni fortemente migliorate. Per ottenere ancora ulteriori miglioramenti, l'intero algoritmo potrebbe essere scritto in assembly (e infatti lo è stato, ma per gli intenti di questo articolo ho deciso di non usarlo). Il motivo principale sta nel fatto che le funzioni *_initStreamRead()* e *_streamRead()* potrebbero risultare molto utili in altri programmi che richiedono la lettura di gruppi di 4 o più pixel alla volta.

Il principio che sta dietro le routine *_stream* è uno di quelli che vengono applicati spesso nell'ottimizzare zone di codice: identificate la parte del programma che viene eseguita più frequentemente e tentate di fare tutte le operazioni di preparazione possibili, prima di entrare in questa sezione. In questo modo, qualsiasi rimozione di operazioni ridondanti dall'interno del ciclo avrà gli effetti più rilevanti sulle prestazioni.

Ringrazio Richard Wilton, autore di "Programmers Guide to PC and PS/2 Video Systems", per la teoria sull'algoritmo di vicinanza di linea.

Listato 1: fastflood.h

```
/*

  fastflood.h

  Fast Flood Fill - FastFlood() routine, by Danny Ross 19th January 1989
  A product of the Johnny Appleseed Software Development Co.
  Requires the assembler support routines from fastpixel.o

*/

#include <exec/types.h>
#include <graphics/rastport.h>
#include <graphics/clip.h>
#include <graphics/gfx.h>

#include <proto/graphics.h>

extern void initStreamRead();
extern int streamRead();

SHORT LineAdjacencyFill();
void scanleft(), scanright();

struct LayersLibrary *LayersBase;
struct RastPort *ff_rp;
int ff_inkPen,ff_outPen;
SHORT ff_xoff, ff_yoff;

void FastFlood(rastp,mode,xp,yp)
  struct RastPort *rastp;
  int mode;
  int xp,yp;
  {
  struct Layer *layer;
  struct ClipRect *cl;

  layer=rastp->Layer;
  cl=layer->ClipRect;
```

```
  ff_xoff=((&(cl->bounds))->MinX);
  ff_yoff=((&(cl->bounds))->MinY);

  if (LayersBase=(struct LayersLibrary *)OpenLibrary("layers.library",0))
    {
    LockLayer(layer->LayerInfo,layer);

    ff_rp=rastp;

    if (mode==0)
      ff_outPen=(int)(ff_rp->AOlPen); else ff_outPen=ReadPixel(ff_rp,xp,yp);

    ff_inkPen=(int)(ff_rp->FgPen);

    (void)LineAdjacencyFill((SHORT)xp,(SHORT)yp,(SHORT)1,(SHORT)xp,(SHORT)xp);

    UnlockLayer(layer);
    CloseLibrary(LayersBase);
    }

  }

SHORT LineAdjacencyFill(seedx,seedy,d,prevxl,prevxr)
  SHORT seedx,seedy,d,prevxl,prevxr;
  {
  SHORT x,y,xl,xr;
  register int v;

  y=seedy; xl=xr=seedx;

  scanleft(&xl,&y); scanright(&xr,&y);

  Move(ff_rp,xl,y); Draw(ff_rp,xr,y);

  for (x=xl; x<=xr; x++)
    {
    v=ReadPixel(ff_rp,x,y+d);
    if ((v!=ff_outPen) && (v!=ff_inkPen)) x=LineAdjacencyFill(x,y+d,d,xl,xr);
    }

  for (x=xl; x<prevxl; x++)
    {
    v=ReadPixel(ff_rp,x,y-d);
    if ((v!=ff_outPen) && (v!=ff_inkPen))   x=LineAdjacencyFill(x,y-d,-d,xl,xr);
    }

  void scanleft(x,y)
    SHORT *x,*y;
    {
    register int v;

    initStreamRead((int)ff_rp,(int)(ff_xoff+(*x)-1),
                                    (int)(ff_yoff+*y),(int)-1);
    do
      {
      --(*x); v=streamRead();
      }
    while ((v!=ff_outPen) && (v!=ff_inkPen));
    ++(*x);
    }
  void scanright(x,y)
    SHORT *x,*y;
    {
    register int v;

    initStreamRead((int)ff_rp,(int)(ff_xoff+(*x)+1),
                                   (int)(ff_yoff+*y),(int)1);
    do
      {
      ++(*x); v=streamRead();
      }
    while ((v!=ff_outPen) && (v!=ff_inkPen));
    --(*x);
    }
```

Listato 2: fastflood.asm

```
HiSoft GenAm 680x0 Macro Assembler v2.08


     1 00.00000000                      ;       OPT     L+              ; produce linkable code.
     2 00.00000000
     3 00.00000000                      ; ---------------------------------------------------------------------------
     4 00.00000000                      ;
     5 00.00000000                      ; Assembler support routines for FastFlood() function
     6 00.00000000                      ;
     7 00.00000000                      ; A product of the Johnny Appleseed Software Development Co.
     8 00.00000000                      ; Written by Danny Ross, 19th January 1989
     9 00.00000000                      ;
    10 00.00000000                      ; C Definitions -
    11 00.00000000                      ;
    12 00.00000000                      ; void initStreamRead(rp,xp,yp,direction)
    13 00.00000000                      ;   struct RastPort *rp;
    14 00.00000000                      ;   int xp,yp;
    15 00.00000000                      ;   int direction;
    16 00.00000000                      ;   {
    17 00.00000000                      ;
    18 00.00000000                      ;   }
    19 00.00000000                      ;
    20 00.00000000                      ; int streamRead();
```

```
21 00.00000000                  ;   {
22 00.00000000                  ;
23 00.00000000                  ;   return(pixelvalue);
24 00.00000000                  ;   }
25 00.00000000                  ;
26 00.00000000                  ; C Example call -
27 00.00000000                  ;
28 00.00000000                  ;   initStreamRead(rp,xp-1,yp,-1);    /* read left */
29 00.00000000                  ;   do
30 00.00000000                  ;     xp--; v=streamRead();
31 00.00000000                  ;   while (v!=border_colour);
32 00.00000000                  ;   xp++;
33 00.00000000                  ;
34 00.00000000                  ; ---------------------------------------------------------------------------
35 00.00000000
36 00.00000000                          INCDIR  ":INCLUDE/"
37 00.00000000
38 00.00000000                          INCLUDE "GRAPHICS/RASTPORT.I"
39 00.00000000                          INCLUDE "GRAPHICS/GFX.I"
40 00.00000000
41 00.00000000                          XDEF    _initStreamRead
42 00.00000000
43   =00000020                  stk     EQU     32                      ; stack offset to reach past saved
44 00.00000000                                                          ; registers
45 00.00000000                  _initStreamRead
46 00.00000000
47 00.00000000 48E73E38                 movem.l D2-D6/A2-A4,-(A7)       ; save registers
48 00.00000004
49 00.00000004 206F0024                 move.l  stk+4(A7),A0            ; A0->RastPort
50 00.00000008 20680004                 move.l  rp_BitMap(A0),A0        ; A0->BitMap
51 00.0000000C
52 00.0000000C 7200                     moveq   #0,D1                   ; clean up D1
53 00.0000000E 32280000                 move.w  bm_BytesPerRow(A0),D1
54 00.00000012 C2EF002E                 mulu.w  stk+14(A7),D1           ; D1=yp*BytesPerRow
55 00.00000016
56 00.00000016 202F0028                 move.l  stk+8(A7),D0            ; D0=xpos
57 00.0000001A 2400                     move.l  D0,D2                   ; take a copy of this xp
58 00.0000001C
59 00.0000001C E648                     lsr.w   #3,D0                   ; convert xp to byte offset
60 00.0000001E 0240FFFE                 andi.w  #$FFFE,D0               ; but make it word aligned
61 00.00000022 D240                     add.w   D0,D1                   ; D1=bitplane offset of xp,yp
62 00.00000024
63 00.00000024 7000                     moveq   #0,D0                   ; clean up D0
64 00.00000026 10280005                 move.b  bm_Depth(A0),D0         ; D0=# of bitplanes
65 00.0000002A 2C00                     move.l  D0,D6                   ; store this for later
66 00.0000002C
67 00.0000002C 45FA0104                 lea.l   storeW(PC),A2           ; A2->storage table for data words
68 00.00000030 43FA00E0                 lea.l   storeP(PC),A1           ; A1->storage table for plane ptrs
69 00.00000034 41E80008                 lea.l   bm_Planes(A0),A0        ; A0->rastport's table of plane ptrs
70 00.00000038
71 00.00000038 E548                     lsl.w   #2,D0                   ; D0=4*(# of bitplanes)
72 00.0000003A D1C0                     adda.l  D0,A0                   ; A0->end of plane ptrs. This is
73 00.0000003C                                                          ; done so we can go thru the ptrs
74 00.0000003C                                                          ; in reverse (see text)
75 00.0000003C 3006                     move.w  D6,D0                   ; D0 is back to normal
76 00.0000003E
77 00.0000003E 3802                     move.w  D2,D4                   ; take a copy of xp
78 00.00000040 0244000F                 andi.w  #$F,D4                  ; calculate pixel number
79 00.00000044
80 00.00000044 262F0030                 move.l  stk+16(A7),D3           ; D3=direction
81 00.00000048 6A04                     bpl.s   initLp                  ; if it's +ve we don't do anything
82 00.0000004A
83 00.0000004A 0A44000F                 eor.w   #15,D4                  ; else reverse 0..15->15..0
84 00.0000004E
85 00.0000004E 2660             initLp  move.l  -(A0),A3                ; pick up a plane pointer
86 00.00000050 D7C1                     adda.l  D1,A3                   ; A3->(xp,yp) address in bitplane
87 00.00000052 22CB                     move.l  A3,(A1)+                ; store plane ptr in table
88 00.00000054 3A13                     move.w  (A3),D5                 ; read screen word
```

```
 89 00.00000056 4A44                    tst.w   D4                      ; do we have to shift it ?
 90 00.00000058 670A                    beq.s   noShift                 ; nope.
 91 00.0000005A 4A43                    tst.w   D3                      ; yes, so which direction ?
 92 00.0000005C 6A04                    bpl.s   goRight                 ; +ve (+1) which means Right
 93 00.0000005E
 94 00.0000005E E86D                    lsr.w   D4,D5                   ; initialise for left scan
 95 00.00000060 6002                    bra.s   noShift
 96 00.00000062
 97 00.00000062 E96D            goRight lsl.w   D4,D5                   ; initialise for right scan
 98 00.00000064
 99 00.00000064 34C5            noShift move.w  D5,(A2)+                ; store data word in table
100 00.00000066 5300                    subq.b  #1,D0                   ; repeat for each bitplane
101 00.00000068 66E4                    bne.s   initLp
102 00.0000006A
103 00.0000006A 4A43                    tst.w   D3                      ; check direction again
104 00.0000006C 6A0C                    bpl.s   bitRht                  ; we are heading right...
105 00.0000006E
106 00.0000006E 303C8000                move.w  #$8000,D0               ; prepare bit marker
107 00.00000072 4A44                    tst.w   D4                      ; do we have to shift it ?
108 00.00000074 670C                    beq.s   initFin                 ; no
109 00.00000076 E868                    lsr.w   D4,D0                   ; correct for mid-word data
110 00.00000078 6008                    bra.s   initFin
111 00.0000007A
112 00.0000007A 7001            bitRht  moveq   #1,D0                   ; prepare bit marker
113 00.0000007C 4A44                    tst.w   D4                      ; do we have to shift it ?
114 00.0000007E 6702                    beq.s   initFin                 ; no
115 00.00000080 E968                    lsl.w   D4,D0                   ; correct for mid-word data
116 00.00000082
117 00.00000082 3480            initFin move.w  D0,(A2)                 ; store bit marker
118 00.00000084
119 00.00000084 5346                    subq.w  #1,D6                   ; D6=# of bitplanes (-1 for DBF)
120 00.00000086 33C600000144            move.w  D6,storDep              ; record the bitmap depth
121 00.0000008C 33C300000146            move.w  D3,storDir              ; record the scan direction
122 00.00000092
123 00.00000092 4CDF1C7C                movem.l (A7)+,D2-D6/A2-A4       ; restore registers
124 00.00000096 4E75                    rts                             ; return to C
125 00.00000098
126 00.00000098                 ;
127 00.00000098                 ; The initialisation routine is finished.
128 00.00000098                 ;
129 00.00000098
130 00.00000098                         XDEF    _streamRead
131 00.00000098                 ;
132 00.00000098                 ;
133 00.00000098                 ; The streamRead() function returns the value of the current pixel and
134 00.00000098                 ; then moves on to the next, reloading data words if a word boundry is
135 00.00000098                 ; reached.
136 00.00000098                 ;
137 00.00000098
138 00.00000098                 _streamRead
139 00.00000098
140 00.00000098 4A7900000146            tst.w   storDir                 ; which way ?
141 00.0000009E 6B3C                    bmi.s   readL                   ; left
142 00.000000A0
143 00.000000A0                 ;
144 00.000000A0                 ; Fast Read Right
145 00.000000A0                 ;
146 00.000000A0
147 00.000000A0 2F0B                    move.l  A3,-(A7)                ; save A3
148 00.000000A2
149 00.000000A2 323A00A0                move.w  storDep(PC),D1          ; initialise D1 as # of planes
150 00.000000A6
151 00.000000A6 41FA008A                lea.l   storeW(PC),A0           ; initialise A0 as -> to data words
152 00.000000AA 7000                    moveq   #0,D0                   ; set colour to 0
153 00.000000AC
154 00.000000AC                 ;
155 00.000000AC                 ; The next section of code reads the colour at the current pixel
156 00.000000AC                 ; and checks to see if we have hit a border. If not we move on to the
157 00.000000AC                 ; next pixel, reloading the data tables if neccesary.
158 00.000000AC                 ;
159 00.000000AC
```

```
160 00.000000AC E5D8            rightPL roxl.w  (A0)+              ; take out bit
161 00.000000AE E350                    roxl.w  #1,D0              ; and shift it into colour
162 00.000000B0 51C9FFFA                dbf.w   D1,rightPL         ; again for the next plane
163 00.000000B4
164 00.000000B4 E3D0                    lsl.w   (A0)               ; shift current bit marker
165 00.000000B6 6620                    bne.s   skipRR             ; not zero so don't reload
166 00.000000B8
167 00.000000B8 41FA0058                lea.l   storeP(PC),A0      ; prepare to get the next word
168 00.000000BC 43FA0074                lea.l   storeW(PC),A1      ; ... of bitplane data
169 00.000000C0 323A0082                move.w  storDep(PC),D1     ; D1=# of bitplanes
170 00.000000C4
171 00.000000C4 2650            resetR  move.l  (A0),A3            ; A3->current address in plane
172 00.000000C6 D7FC00000002            adda.l  #2,A3              ; move on to next word
173 00.000000CC 32D3                    move.w  (A3),(A1)+         ; read next word and store in table
174 00.000000CE 20CB                    move.l  A3,(A0)+           ; store bitplane address
175 00.000000D0 51C9FFF2                dbf.w   D1,resetR          ; and repeat for all planes
176 00.000000D4
177 00.000000D4 32BC0001                move.w  #$0001,(A1)        ; reset bit marker
178 00.000000D8
179 00.000000D8 265F            skipRR  move.l  (A7)+,A3           ; restore A3
180 00.000000DA 4E75                    rts
181 00.000000DC
182 00.000000DC
183 00.000000DC                 ;
184 00.000000DC                 ;Fast Read Left
185 00.000000DC                 ;
186 00.000000DC
187 00.000000DC 2F0B            readL   move.l  A3,-(A7)           ; save A3
188 00.000000DE
189 00.000000DE 323A0064                move.w  storDep(PC),D1     ; initialise D1 as # of planes
190 00.000000E2
191 00.000000E2 41FA004E                lea.l   storeW(PC),A0      ; initialise A0 as -> to data words
192 00.000000E6 7000                    moveq   #0,D0              ; set colour to 0
193 00.000000E8
194 00.000000E8                 ;
195 00.000000E8                 ; The next section of code reads the colour at the current pixel
196 00.000000E8                 ; and checks to see if we have hit a border. If not we move on to the
197 00.000000E8                 ; next pixel, reloading the data tables if neccesary.
198 00.000000E8                 ;
199 00.000000E8
200 00.000000E8 E4D8            leftPL  roxr.w  (A0)+              ; take out bit
201 00.000000EA E350                    roxl.w  #1,D0              ; and shift it into colour
202 00.000000EC 51C9FFFA                dbf.w   D1,leftPL          ; again for the next plane
203 00.000000F0
204 00.000000F0 E2D0                    lsr.w   (A0)               ; shift current bit marker
205 00.000000F2 661A                    bne.s   skipLR             ; not zero so don't reload
206 00.000000F4
207 00.000000F4 41FA001C                lea.l   storeP(PC),A0      ; prepare to get the next word
208 00.000000F8 43FA0038                lea.l   storeW(PC),A1      ; ... of bitplane data
209 00.000000FC 323A0046                move.w  storDep(PC),D1     ; D1=# of bitplanes
210 00.00000100
211 00.00000100 2650            resetL  move.l  (A0),A3            ; A3->current address in plane
212 00.00000102 32E3                    move.w  -(A3),(A1)+        ; read next word and store in table
213 00.00000104 20CB                    move.l  A3,(A0)+           ; store bitplane address
214 00.00000106 51C9FFF8                dbf.w   D1,resetL          ; and repeat for all planes
215 00.0000010A
216 00.0000010A 32BC8000                move.w  #$8000,(A1)        ; reset bit marker
217 00.0000010E
218 00.0000010E 265F            skipLR  move.l  (A7)+,A3           ; restore A3
219 00.00000110 4E75                    rts
220 00.00000112
221 00.00000112                 ;
222 00.00000112                 ; Working data initialised by initStreamRead().
223 00.00000112                 ;
224 00.00000112
225 00.00000132                 storeP  ds.l    8                  ; plane pointers
226 00.00000142                 storeW  ds.w    8                  ; current data words
227 00.00000142
228 00.00000144                 storeB  ds.w    1                  ; bit marker (possibly)
229 00.00000144
```

```
230 00.00000146                    storDep ds.w    1                   ; depth of bitmap
231 00.00000148                    storDir ds.w    1                   ; direction
232 00.00000148
233 00.00000148                            end
```

Listato 3: demo.c

```
/*

    Fast Fill routine - Demonstration program by Danny Ross.

    A product of the Johnny Appleseed Software Development Co.
    Written using CygnusED Professional, Compiled with Lattice C version 4.0
    Requires fastflood.h

*/

#include <stdio.h>

#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <graphics/gfx.h>
#include <graphics/gfxmacros.h>

#include "df1:fastflood.h"

struct TextAttr MyFont =
    {
    "topaz.font",
    TOPAZ_EIGHTY,
    FS_NORMAL,
    FPF_ROMFONT,
    } ;

#define gwi 120
#define ghi 20

#define FLOOD 0
#define FASTFLOOD 1

SHORT Outline1[] = { 0,2,2,0,119,0,121,2,121,17,119,19,2,19,0,17,0,2 } ;
SHORT Outline2[] = { 2,4,4,2,117,2,119,4,119,15,117,17,4,17,2,15,2,4 } ;

struct IntuiText flood_text = { 15,1,JAM1,5,6,&MyFont,"    Flood",NULL };
struct IntuiText fastflood_text = { 15,1,JAM1,5,6,&MyFont,"  Fast Flood",NULL };
struct IntuiText colup_text = { 15,1,JAM1,5,6,&MyFont,"      +",NULL };
struct IntuiText coldn_text = { 15,1,JAM1,5,6,&MyFont,"      -",NULL };
struct IntuiText reset_text = { 15,1,JAM1,5,6,&MyFont,"  Reset",NULL };
struct IntuiText quit_text = { 15,1,JAM1,5,6,&MyFont,"    Quit",NULL };

struct Border gadget_border2 = { 0,0,15,0,JAM1,9,&Outline2[0],NULL } ;
struct Border gadget_border = { 0,0,8,0,JAM1,9,&Outline1[0],&gadget_border2 } ;

struct Gadget flood_gadget =
    {
    NULL,435,40,gwi,ghi,GADGHCOMP,RELVERIFY,
    BOOLGADGET,(APTR)&gadget_border,NULL,&flood_text,0x0000,NULL,0,NULL
    } ;

struct Gadget fastflood_gadget =
    {
    &flood_gadget,435,70,gwi,ghi,GADGHCOMP,RELVERIFY,
    BOOLGADGET,(APTR)&gadget_border,NULL,&fastflood_text,0x0000,NULL,1,NULL
    } ;
```

```
struct Gadget colup_gadget =
    {
    &fastflood_gadget,370,120,gwi,ghi,GADGHCOMP,RELVERIFY,
    BOOLGADGET,(APTR)&gadget_border,NULL,&colup_text,0x0000,NULL,2,NULL
    } ;

struct Gadget coldn_gadget =
    {
    &colup_gadget,500,120,gwi,ghi,GADGHCOMP,RELVERIFY.
    BOOLGADGET,(APTR)&gadget_border,NULL,&coldn_text,0x0000,NULL,3,NULL
    } ;

struct Gadget reset_gadget =
    {
    &coldn_gadget,370,150,gwi,ghi,GADGHCOMP,RELVERIFY,
    BOOLGADGET,(APTR)&gadget_border,NULL,&reset_text,0x0000,NULL,4,NULL
    } ;

struct Gadget quit_gadget =
    {
    &reset_gadget,500,150,gwi,ghi,GADGHCOMP,RELVERIFY,
    BOOLGADGET,(APTR)&gadget_border,NULL,&quit_text,0x0000,NULL,5,NULL
    } ;

struct NewScreen ns =
    {
    0,0,640,250,4,0,1,HIRES,CUSTOMSCREEN,&MyFont,
    "Flood Fill Demonstration Screen",NULL,NULL
    } ;

struct NewWindow nw =
    {
    0,11,640,239,0,1,CLOSEWINDOW|GADGETUP|MOUSEBUTTONS,
    WINDOWCLOSE|WINDOWDRAG|NOCAREREFRESH|RMBTRAP|ACTIVATE,&quit_gadget,NULL,
    "By Danny Ross, 10th March 1989",NULL,NULL,0,0,0,0,CUSTOMSCREEN
    } ;

struct RastPort *rp;
struct Screen *scrn = NULL;
struct Window *wndo = NULL;
struct ViewPort *vp;

int bpen,ipen,col=1;
int fillStyle=FLOOD;

struct IntuitionBase *IntuitionBase = NULL;
struct GfxBase *GfxBase = NULL;

void initialise(), abEnd(), setfillstyle();

void initialise()
    {
    if ((IntuitionBase=(struct IntuitionBase*)OpenLibrary("intuition.library",0))==NULL) abEnd("Can't open Intuition Library");
    if ((GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",0))==NULL) abEnd("Can't open Graphics Library");
    if ((scrn=(struct Screen *)OpenScreen(&ns))==NULL) abEnd("Can't open Screen");

    nw.Screen=scrn;

    if ((wndo=(struct Window *)OpenWindow(&nw))==NULL) abEnd("Can't open Window");
    vp=(struct ViewPort *)ViewPortAddress(wndo);
    rp=wndo->RPort;

    SetSoftStyle(rp,FSF_ITALIC,255);

    SetAPen(rp,15); RectFill(rp,20,30,320,230);
    SetAPen(rp,0); RectFill(rp,21,31,319,229);
    SetAPen(rp,col); Move(rp,445,109); Text(rp,"Fill Colour",11);
    SetAPen(rp,15); bpen=15; SetOPen(rp,15);

    setfillstyle(FLOOD);
```

```
void abEnd(s)
    char *s;
    {
    if (s) printf("%s\n",s);
    if (scrn) CloseScreen(scrn);
    if (wndo) CloseWindow(wndo);
    if (GfxBase) CloseLibrary(GfxBase);
    if (IntuitionBase) CloseLibrary(IntuitionBase);
    exit();
    }

void floodfill(fillStyle,col,xp,yp)
    int col,xp,yp;
    {
    SetAPen(rp,col); ipen=col;
    switch (fillStyle)
        {
        case FLOOD : { Flood(rp,0,xp,yp); break; }
        case FASTFLOOD : { FastFlood(rp,0,xp,yp); break; }
        default : { break; }
        }
    SetAPen(rp,15);
    }

main()
    {
    struct IntuiMessage *message;

    int mx,my,quit,lost;
    ULONG class;
    USHORT code;

    struct Gadget *gadget;

    initialise();

    quit=FALSE;
    while (quit==FALSE)
        {
        Wait( 1<< (wndo->UserPort->mp_SigBit));
        while ((message=(struct IntuiMessage *)GetMsg(wndo->UserPort))!=NULL)
            {
            class=message->Class; code=message->Code;
            mx=message->MouseX; my=message->MouseY;
            gadget=(struct Gadget *)message->IAddress;

            ReplyMsg(message);
            switch (class)
                {
                case CLOSEWINDOW :
                    {
                    quit=TRUE;
                    break;
                    }
                case GADGETUP :
                    {
                    switch (gadget->GadgetID)
                        {
                        case 0 : { setfillstyle(FLOOD); break; }
                        case 1 : { setfillstyle(FASTFLOOD); break; }
                        case 2 :
                            {
                            col=((col+1)%15); SetAPen(rp,col);
                            Move(rp,445,109); Text(rp,"Fill Colour",11); SetAPen(rp,15);
                            break;
                            }
```

```
                    case 3 :
                        {
                        if (--col<0) col=14;
                        SetAPen(rp,col);
                        Move(rp,445,109); Text(rp,"Fill Colour",11); SetAPen(rp,15);
                        break;
                        }
                    case 4 :
                        {
                        SetAPen(rp,15); RectFill(rp,20,30,320,230);
                        SetAPen(rp,0); RectFill(rp,21,31,319,229);
                        SetAPen(rp,15);
                        break;
                        }
                    case 5 : { quit=TRUE; break; }
                    }
                break;
                }
            case MOUSEBUTTONS :
                {
                if (code==SELECTDOWN)
                    {
                    lost=TRUE;
                    while ((message=(struct IntuiMessage *)GetMsg(wndo->UserPort))==NULL)
                        {
                        mx=wndo->MouseX; my=wndo->MouseY;
                        if ((mx>=20) && (mx<=320) && (my>=30) && (my<=230))
                            {
                            if (lost) { lost=FALSE; Move(rp,mx,my); }
                            Draw(rp,mx,my);
                            }
                        else
                            {
                            lost=TRUE;
                            }
                        }
                    ReplyMsg(message);
                    }
                else if (code==MENUDOWN)
                    {
                    if ((mx>20) && (mx<320) && (my>=30) && (my<=230))
                        {
                        floodfill(fillStyle,col,mx,my);
                        }
                    }
                break;
                }
            default :
                {
                break;
                }
            }
        }
    }

    abEnd(NULL);
    }

void setfillstyle(style)
    int style;
    {
    Move(rp,380,29);
    fillStyle=style;
    switch (fillStyle)
        {
        case FLOOD : { Text(rp,"Flood Algorithm - Flood()    ",29); break; }
        case FASTFLOOD : { Text(rp,"Flood Algorithm - FastFlood()",29); break; }
        }
    }
```

I dischi di Fred Fish - 3

Prima guida ragionata al loro contenuto

UTILITY PER IL VIDEO

86 AUTOPOINT			SELEZIONA WINDOW SOTTO PUNTATORE AUTOMAT.+SALVA SCHERMO	[KATSCH]
112 BULLY		*	MUOVE LO SCHERMO (PER MULTI-DEMOS)	[MEYER]
86 CLICKTOFONT	1.1	*	PORTA LA WINDOW DAVANTI FACENDO UN DOPPIO CLICK SU ESSA	[NESBITT]
94 CLICKUPFRONT	1.0	*	PORTA DELLE WINDOW DAVANTI FACENDO 2 CLICK SU DI ESSE	[CERVONE]
65 CLOSEWB		*	CHIUDE SCHERMO CORRENTE DEL WB E APRE SU QUELLO IN USO	[DILLON]
32 COL	1.1		CAMBIA TRA LE 60 E 80 COLONNE	[BEOGELEIN]
73 EXPOSE		*	ACCESSO A TUTTE LE DRAG BAR DELLO SCHERMO	[RUSSEL]
94 HELIOSMOUSE	1.0	*	ATTIVA AUTOMATICAMENTE LE WINDOW DOVE C'E' IL MOUSE	[CERVONE]
111 HELIOSMOUSE	1.1	*	ATTIVA AUTOMATICAMENTE LE WINDOW DOVE C'E' IL MOUSE	[CERVONE]
49 LACETOGL		*	ATTIVA/DISATTIVA L'INTERLACCIAMENTO	[RETHEMEYER]
70 LENS	1.0		INGRANDISCE PICCOLA AREA DI SCHERMO ATTORNO AL PUNTATORE	[KONZ]
130 MACH	1.6A	*	ACCELLERA IL MOUSE WITH HOTKEY E CLOCK/TIMER ECC.	[MOATS]
54 MOREROWS			INCREMENTA L'ALTEZZA E LA LUNGHEZZA DELLO SCHERMO	[KATIN/ MACKRANZ]
73 MOUSEOFF		*	INIBISCE PUNTATORE MOUSE DOPO 10 SECONDI DI NON USO	[JENKINS]
75 MOUSEOFF (JENKINS)]		*	INIBISCE PUNTATORE MOUSE DOPO 10 SECONDI DI NON USO	[SMYTHE
87 MOVEPOINTER		*	MUOVE IL PUNTATORE IN UNA SPECIFICATA POSIZIONE SCHERMO	[CEWY]
87 MOVEWINDOW		*	MUOVE WINDOW (E RIDIMENSIONA) IN UNA LOCAZIONE SCHERMO	[CEWY]
133 OVERSCAN		*	CREA FINESTRE IN OVERSCAN PAL	[FREUND/DOY]
55 PALETTE	1.1	*	CAMBIA COLORE SULLO SCHERMO	[SCHEPPNER (HEATH)]
43 POPCOLOURS	1.0	*	CAMBIA COLORI DELLO SCHERMO CON GADGET SLIDER	[ZAMARA/ SULLIVAN]
79 SCNSIZER		*	CAMBIA DIMENSIONE ALLO SCHERMO (IN PREFERENCES)	[FLORYAN]
33 SCREENDUMP		*	FA UN DUMP DELLO SCHERMO PIU' ALTO ALLA STAMPANTE	[SCHEPPNER]
27 SCREENPRINT		*	UTILITY PER DUMP SCHERMO (AMIGABASIC)	[SCHEPPNER]
55 SCREENSAVE		*	SALVA SCHERMO (INC HAM) COME UN FILE IFF CON ICON	[SCHEPPNER]
89 SCREENSHIFT	1.0	*	PERMETTE ALLO SCHERMO DI MUOVERSI ATTORNO AL PREFERENCES	[MAH]
18 SCRIMPER		*	STAMPA SCHERMO - WB O CLI	[KIVOLOWITZ]
9 SETLACE		*	ACCENDE/SPEGNE IL MODO INTERLACCIATO	[PARISEAU]
79 SETLACE		*	ACCENDE/SPEGNE IN MODO INTERLACCIATO (CODICE MOLTO PIC.)	[NESBITT]
66 SNAPSHOT	1.0		DUMP DELLO SCHERMO AD UN FILE IFF-ATTIVATO CON CTRL-ESC	[ROUAIX]
73 SNAPSHOT	2.0		DAMP DELLO SCHERMO AD UN FILE IFF-ATTAVATO CON CTRL-ESC	[RAUAIX]
65 SUNMOUSE	1.0	*	ATTIVA WINDOW SOTTO IL PUNTATORE DEL MOUSE	[EVERDEN]
70 VPG	1.0		GENERA TEST PATTERNS PER MONITORS	[BERRO]
87 WARPTEXT		*	NUOVA VERSIONE SUL FISH96	[KELLY]
96 WARPTEXT	2.0	*	ROUTINE PER RENDERE IL TESTO SULLO SCHERMO MOLTO VELOCE	[KELLY]
128 WKEYS		*	PERMETTE ALLA TASTIERA DI CREARE E MANIPOLARE WINDOW	[CERVONE]
107 XTEXT	1.0	*	ROUTINE CHE PERMETTE DI SCRIVERE TESTO SU SCHERMO VELOC.	[MICAL]

UTILITY PER IL SISTEMA

13 AMICRON	2,3	*	LISTA TASK IN ESECUZIONE IN SPECIFICATO TEMPO E DATA	[SCHAEFFER]
79 ASSIGNDEV		*	PERMETTE + NOMI PER SINGOLI DEVICE	[LINDSAY]
111 ASSIGDEV		*	PERMETTE + NOMI PER SINGOLI DEVICE	[LINDSAY]
61 ATPATCH		*	TRASFORMATORE DI PATCH PER LAVORARE CON 1.2	[STAUB]
81 AUTOFACC			RIDUCE E MUOVE DIETRO LA FINESTRA CREATA DA FACC ADSG	[RACHMAT]
69 BLITLAB	1.1	*	PERMETTE DI CIMENTARSI IN ESPERIMENTI CON IL BLITTER	[ROKICKI]
84 BLITLAB	1.2	*	PERMETTE DI CIMENTARSI IN ESPERIMENTI CON IL BLITTER	[ROCKIKI]
79 CMD	1	*	RIDIRIGE L'OUTPUT SERIALE/PARALLELO A UN FILE	[SHEPPNER]
86 CMD	3	*	RIDIRIGE L'OUTPUT SERIALE/PARALLELO A UN FILE	[SHEPPNER]
95 CMD	4	*	RIDIRIGE L'OUTPUT SERIALE/PARALLELO A UN FILE	[SHEPPNER]

43	COPPER	*	DISASSEMBLA UNA LISTA DI ISTRUZIONI PER IL COPPER	[EVERDEN]
66	FREE	*	RITORNA I BYTES DISPONIBILI DI UN SPECIFICATO DEVICE	[SMYTHE]
105	FLAMKEY 1.0		PERMETTE PASSWORD USANDO UNA PICCOLA CHIAVE NEL MENU BAR	[LIVSHITS]
95	GOMF 1.0		GUARDA LE GURU ED AIUTA AD USCIRNE CORRETTAMENTE	[JOHNSEN]
95	JOURNAL 1.0	*	REGISTRA LE SEQUENZE DI EVENTI DI MOUSE/TASTIERA	[CERVONE]
81	KEYLOCK		CONGELA TASTIERA E MOUSE FINCHE' SI INTRODUCE PASSWORD	[RACHMAT]
36	KICKBENCH		METTE KICKSTART(1.1/1.2P)E WBENCH SULLO STESSO DISCO	[GARIEPY]
79	LOADIT	*	PERMETTE AD UN FILE DI ESSERE CARICATO E TENUTO	[NESBIT]
55	PIPEDEVICE	*	PERMETTE O/P DI PROGRAMMI USATI COME I/P DI UN ALTRO	[DILLON]
84	PIPEHANDLER	*	GESTISCE IL PIPE	[PUCKETT]
32	PREF	*	SETTA IL PREFERENCES	
95	PRINTERSTEALER	*	DEVIA L'OUTPUT DELLA STAMPANTE AD UN FILE	[LIVSHITS (FORGEAS)]
26	PS		VISUALIZZA/SETTA LA PRIORITA' DEI VARI TASK	[WILLIAMS]
95	RECORD-REPLEY		REGISTRA(E VA RIVEDERE)TUTTI GLI EVENTI TASTIERA/MOUSE	
105	RECORD-REPLEY 2.0		REGISTRA(E FA RIVEDERE)TUTTI GLI EVENTI TASTIERA/MOUSE	
20	SETMOUSE2	*	SETTA IL MOUSE PER ESSERE USATO NELL'ALTRA PORTA	[BURNS]
107	YOYO	*	MOUVE SCHERMO SU E GIU' PER TEST MEMORIA TRASHING	[VERMEULEN]

UTILITY PER LA MEMORIA

105	ADDLICKMEM	*	RENDE DISPONIBILE LA RAM DEL KICKSTART(SE KS IN ROM!)	[MCDIARMID]
58	ASDG-RRD 3.0		RAM DISK (VD0:)LA QUALE RESISTE AL WARM BOOT	[KIVOLOWITS]
58	CLEANRAMDISK		PULISCE LE LOCAZIONI DI MEMORIA DI ASDG RAM DISK	[KIVOIOWITS]
105	CLEAR		FILES INUSATI IN RAM CON VALORI SPECIFICI	[MCDIARMID]
58	DELETERAMDISK		RIMUOVE ASDG RAM DISK AL PROSSIMO REBOOT	[KIVOIOWITS]
58	FASTMEM	*	ATTIVA/DISATTIVA LA FAST MEM	[KIVOLOWITZ]
107	FRAGIT	*	DELIBERA FRAMMENTI DI MEMORIA CHIP PER TESTING	[VERMEULEN]
69	FRAGS	*	VISUALIZZA LA FRAMMENTAZIONE DELLA MEMORIA LIBERA	[[MEYER]
85	HIDE		FORZA L'ALLOCAZIONE DI MEMORIA NELLA MEMORIA CHIP	[ROUAIX]
66	MAILOCTEST	*	ALLOCA E TESTA LA MEMORIA LIBERA	[WEBER]
58	MEMCLEAR	*	AZZERA TUTTA LA MEMORIA LIBERA	[HODGSON]
107	MEMLIST	*	RIPORTA DISPONIBILI FRAMMENTI,LOCAZIONI E DIMENSIONE MEM	[VERMEULEN]
33	MEMORYVIEW	*	CREA UNA WINDON NELLA MEMORIA	[SCHWAB]
48	MEMWATCH	*	PROTEGGE CONTRO IL TRASHING DI BASSA MEMORIA	[TOEBES]
87	MEMWATCH II	*	CONTROLLA PER IL TRASHING DI BASSA MEMORIA	[TOEBES]
56	MEMMERGE	*	TENTA DI UNIRE LA MEMORIA ESTERNA IN UN BLOCCO	[SCHEPPNER]
95	MEMMERGE II	*	TENTA DI ALLOCARE LA MEMORIA ATTRAVERSO I LIMITI BORDO	[SHEPPNER]
31	RAMSPEED	*	MISURA LA VELOCITA' DELLA MEMORIA INTERNA ED ESTERNA	[KIVOLOWITS]
96	TIMERAM		TEST SULLA VELOCITA' DELLA FAST E CHIP RAM	[TAKAHASHI]
131	WFRAGS	*	MOSTRA COSTANTEMENTE LA FRAMMENTAZIONE DELLA MEMORIA	[ROKICKI]

VIRUS

126	VCHECK 1,2		CONTROLLA LA PRESENZA DI VIRUS (CONTROLLA RAM E DISCHI)	[KOESTER]
137	VIRUSX	*	CONTROLLA OGNI DISCO INSERITO PER I VIRUS	[TIBBETT]

VOCE

1	SPEECH 1	*	SEMPLICE DEMO PARLATO - RIDUCE SPEECHTOY	[PECK]
5	SPEECH.DEMO 1.1	*	SEMPLICE DEMO PARLATO - RIDUCE SPEECHTOY	[PECK]
1	SPEECHTOY	*	PROGRAMMA DEMO PARLATO CON FACCIA	[LUCAS]
5	SPEECHTOY 1.1	*	PROGRAMMA DEMO PARLATO CON FACCIA E ICON	[LUCAS]
46	VALSPEACK	*	TRADUZIONE DELL'INPUT STANDARD PER VALLEY SPEECH	

WORKBENCH

59	DROPCLOTH 1.0		SOSTITUISCE LO SCHERMO DE WB CON PATTERN	[LAVITSKY/ KIVOIOWITS]
128	DROPCLOTH 2.2		METTE FIGURA SUL WB BACKDROP (INC FIGURE)	[LAVITSKY]
59	DROPSHADOW 1.1		CREA OMBRE PER LE WINDOW DEL WB	[MACKRANZ]
74	DROPSHADOW 2.0		CREA OMBRE PER LE WINDOW DEL WB	[MACKRANZ]
87	DROPSHADOW 2.0	*	CREA OMBRE PER LE WINDOW DEL WB	[MACKRANZ]
112	DROPSHADOW 2.0		CREA OMBRE PER LE WINDOW DEL WB	[MACKRANZ]
59	FIXWB	*	FISSA LO SCHERMO DEL WB E LO SOSTITUISCE CON UN PATTERN	[SCHWAB]
96	HACKBENCH	*	WORKBENCH-COME PROGRAMMA PER ESPERIMENTI	[KINNERSLEY]
65	MWB 1.01	*	CREA UN NUOVO SCHERMO WB	[DILLON]
31	TICON	*	VISUALIZZA FILE TESTO DA UN ICONA	[GOODEVE]
71	TIMESET 0.1		PERMETTE DI SETTARE L'ORA DAL WORKBENCH	[DEH]
31	XICON	*	PERMETTE AD UN ICONA DI ESEGUIRE UN FILE COMANDO CLI	[GOODEVE]

WORKBENCH UTILITY

121 WBCOLORS	1.0	*	SETTA I COLORI DA UN PREDETERMINATO SET	[LINDHAL]

AGENDA

34 CALENDAR		*	VISUALIZZA MENSILMENTE APPUNTAMENTI DA CALENDARIO	[LEIVIAN]
32 CALENDAR	1.1	*	CALENDARIO/DIARIO (AMIGABASIC)	[HURST]

ARCHIVIO FILE

40 ARC	0.16		PROGRAMMA ARCHIVIO STANDARD DE FACTO	[BRAND]
70 ARC	0.23		PROGRAMMA ARCHIVIO STANDARD DE FACTO	[BRAND]
26 ARCHX		*	ARCHIVIA DIVERSI FILES DI TESTO IN UN SINGOLO FILE	[MINOW]
53 ARCRE		*	CREA SCRITTURE RINOMINATE PER ARC FILES	[HOFFMAN]
48 BRU	ALPHA1		HARD-DISK E/O ARCHIVIATORE DI FILE	[FISH]
6 COMPRESS		*	COMPRIME FILES USANDO ADAP.LEMPEL-ZIF CODING	[SPENCER]
51 COMPRESS	1.1	*	COMPATTA FILES USANDO ADAP. LAMPEL-ZIF CODING	[SPENCER]
87 FIZ	1.0	*	USATO PER ESTRARRE FILES DA ZOO ARC DANNEGGIATI	[DHESI/WATERS]
103 PACK-IT			COMPRIME TUTTI I FILES E DIRECTORY IN UN UNICO FILE	[KEMPER]
2 PORTAL		*	USATO PER METTERE IN FILA FILES TESTO IN UN UNICO FILE	[MINOW/FISH]
28 SHAR		*	COMPATTA E DECOMPATTA ARCHIVI (ARCHIVIATORE SHELL)	[WECKER]
92 SHAR		*	COMPATTA E DECOMPATTA ARCHIVI (ARCHIVIATORE SHELL)	[DUFOE]
10 SQ	3.2	*	COMPATTA FILE DA CP/M - LENTO	[SCHAEFFER (GREENLAW)]
51 SQ	3.3	*	COMPATTA FILE DAL MONDO CP/M	[GREENLAW/ SWAN]
53 TARSPLIT		*	ESTRAE FILES DA ARCHIVI IN UNIX TAR	[JONES/MEYER]
50 UNIXARC		*	UNIX SYS V PROGRAMMI COMPATIVILI ARC	[SYSTEM ENHANCE ASS.]
10 USQ	3.2	*	ESPANSIONE FILE DA CP/M - LENTO	[SCHAEFFER (GREENIAW)]
51 USQ	3.3	*	ESPANSIONE FILE DAL MONDO CP/M	
87 ZOO	1.42A		ARCHIVIO FILE - SIMILE AD ARC	[DHESI/WATERS]
108 ZOO	1.42B		ARCHIVIO FILE - SIMILE AD ARC	[DHESI/WATERS]
136 ZOO	1.71		ARCHIVIO FILE - SIMILE AD ARC	[WATERS (DHESI)]

COLLEZIONE DI ICONE

44 ICONS			COLLEZIONE DI 11 ICONE	
67 ICONS			COLLEZIONE DI 14 ICONE	[WHITE
71 ICONS			COLLEZIONE DI ICONE-MACCHINA DA SCRIVERE,VT100 E MUSICA	
82 ICONS			COLLEZIONE DI 52 ICONE	
124 ICONS			COLLEZIONE DI 28 ICONE ANIMATE	[PFOST]
137 JEANSICONS			30 NUOVE ICONE	[JEANS]

DATABASE

13 ADDBOOK.PAS		*	LIBRO INDIRIZZO/AGENDA (ABASIC)	[MILLER]
32 ADDRESS	1.0	*	LIBRO INDIRIZZO/AGENDA (AMIGABASIC)	[HURST]
70 BLACKBOOK		*	NOMI E INDIRIZZI	[NELSON]
13 CARDFI.BAS		*	SEMPLICISSIMO DATABASE (ABASIC)	

DATI DELLE DIRECTORY DEI DISCHI

89 AMICUS.DBF			INDICE AMICUS PER DIRMASTER (1-16)	
103 AMICUS.MFF			INDICE AMICUS (1-22) PER MFF MICROFICHE FILER	
130 AMICUS1-22			INDICE AMICUS PER DIRMASTER (1-22)	
89 FISH.DBT			INDICE FISH PER DIRMASTER (DISCHI SELEZIONATI)	
103 FRED_FISH.MFF			INDICE FISH (1-80) PER MFF MICROFICHE FILER	
89 FREDFISH-1-80			INDICE FISH (1-80) PER FILEIISG	
130 MCA			MICRO COMPUTER ASSOCIATION (DIRMASTER)	[PETERS]

DATI PER DATABASE

103	FRED.LBR		INDICE DEI FISH 1-110 (PER LA LIBRERIA)	
103	ORDER_ENTRY.MFF		APPLICAZIONE PER MMF MICROFICHE FILER	

DISEGNANDO

29	AEGISDRAWDEMO	1.00	DEMO DELL'AEGIS DRAW - CON SAVE DISABILITATO	[AEGIS]
117	EXP_DEMO	1.1	DEMO DELLA VERSIONE DI EXPRESS PAINT	[PAR SOFTWARE]
1	FREEDRAW	*	PICCOLO PROGRAMMA PER DISEGNARE	[ROSS]
56	MCAD	1.1.0	PROGRAMMA SPECIFICO PER DISEGNARE	[MOONEY]
59	MCAD	1.2.2	PROGRAMMA SPECIFICO PER DISEGNARE	[MOONEY]
74	MCAD	1.2.4	PROGRAMMA SPECIFICO PER DISEGNARE	[MOONEY]
128	PAINT	*	PROGRAMMA PER DISEGNARE MOLTO SEMPLICE-IN LINGUAGGIO WEB	[LEE]
15	POLYDRAW	1.2	PROGRAMMA PER DISEGNARE (ABASIC)	[ADDISON]
31	VDRAW	1.2	PROGRAMMA PER DISEGNARE-MIGLIORA MOLTO IL DISEGNO LIBERO	[VERMEULEN]
38	VDRAV	1.08	PROGRAMMA PER DISEGNARE-MIGLIORA MOLTO IL DISEGNO LIBERO	[VERMEULEN]
31	VDRAW	1.14	PROGRAMMA PER DISEGNARE-MIGLIORA MOLTO IL DISEGNO LIBERO	[VERMEULEN]
52	VDRAW	1.16	PROGRAMMA PER DISEGNARE-MIGLIORA MOLTO IL DISEGNO LIBERO	[VERMEULEN]
52	VDRAW	1.19	PROGRAMMA PER DISEGNARE-MOLTO MIGLIORATO	[VERMEULEN]

DISK DRIVER

98	HDDRIVER	*	DRIVER PER WD-1002-05 CONTROLLORE HARD DISK	[KENT]

DRIVER PER STAMPANTE

108	DOTS-PERFECT	*	EPSON MX80 CON DOTS-PERFECT INSTALLATO -DRIVER	[AKINS]
60	NECP6		PARAMETRI PER PRINTER DRIVER NEC P6 (PER PRTDRVGEN)	[THOMSEN]
15	OKIDATADUMP	*	OKIDATA ML92 DRIVER SORGENTE - CON WBENCH DUMP	[GLUECKER]
129	PAINTJET		DRIVER UFFICIALE HOWLETT-PACKARD PAINTJET	[HAWLETT-PACKARD]
87	TEK4695		TEKTRONIK 4695 PRINTER DRIVER	[STAUB]
96	TEK4695	*	TEKTRONIK 4695/4696 PRINTER DRIVER	[STAUB]
60	THINKJET		PARAMETRI HP THINK JET DRIVER (PER PRTDRVGEN)	[WEIBLEN]
128	TOSHIBA	*	3 IN UNO TOSHIBA PRINTER DRIVER - MODO QUME	[MARIANI]
60	PRTDRVGEN	1.1	GENERA PRINTER DRIVER GENERICI	[THOMSEN]
80	PRTDRVGEN	2.2B	GENERA PRINTER DRIVER GENERICI	[THOMSEN]
90	PRTDRVGEN	2.2B	GENERA PRINTER DRIVER GENERICI	[THOMSEN]

EDITOR DI FILE

10	FILEZAP	2.1	PER EDITARE FILE IN ASCII/HEX	[HODGSON]
14	FILEZAP	2.2 *	PER EDITARE FILE IN ASCII/HEX	[HODGSON]

ESEMPI DI MODULA2

113	ALERT	2.0 *	MOSTRA COME VISUALIZZARE UN ALLARME IN MODULA 2	
113	FRAGS	2.0 *	MOSTRA FRAMMENRI DI MEMORIA	[SCHAUB]
113	QUEENS	*	IL PROBLEMA DELLE SETTE REGINE	[SCHAUB]
113	SCANLIST	*	SCANSIONE ATTRAVERSO EXEC MOSTRANDO NOMI DEI NODI	[SCHAUB]
113	SIGNALS	2.0 *	DEMO DELL'USO DI UN SEGNALE IN MODULA 2	[SHAUB]

FINANZA

120	BANKN	1.3	COMPLETO LIBRO DI CONTROLLO PER SISTEMI DI BILANCIO	[CARTER]
43	BBM		AFFARI MENAGERIALI - RIDUCE LUNGHEZZA FILE DEMO	[BEST S/W]
74	FUNDS	1.1 *	SEGUE I FONDI MUTUI E PROVVEDE AI PREZZI	[STRACK]
59	SUPERMORT	0.5	CALCOLO AMMORTAMENTO	[SCHRETLEN]

FONT

34	APPLE		16 LINNE	[FISCHER]
34	ASTRA		16 LINEE	[FISCHER]

81	BASIC		CARATTERI 28 E 32 LINEE - HELVETICA-STILE PER TITOLI	
34	BOLDTYPE		16 LINEE	[FISCHER]
34	BROADWAY		16 LINEE	[FISCHER]
81	BUBBLE		14 18 E 22 LINEE - CARATTERI APERTI	
34	BYTE		8 LINEE	[FISCHER]
34	COLLOSAL		8 LINEE	[FISCHER]
34	COUNT		8 LINEE	[FISCHER]
34	COUNTDOWN		16 LINEE	[FISCHER]
34	CYBER		16 LINEE	[FISCHER]
81	ERICS		8 LINEE - SANS-SERIF STILE (ANCHE UNA VERSIONE BOLD)	
34	EXPANDED		16 LINEE	[FISCHER]
34	FLOW		8 LINEE	[FISCHER]
34	FRANKFURTER		16 LINEE	[FISCHER]
34	GOTHIC		8 LINEE	[FISCHER]
71	IBM5		EMULA CARATTERI IBM PC	[KITTEL]
112	LONGISLAND		12 18 LINEE - CARATTERI NON PROPORZIONALI	
34	NINETYS		16 LINEE	[FISCHER]
34	OUTLINE		8 E 16 LINEE	[FISCHER]
61	PEARLFONT		8 LINEE - CARATTERI CLEAN,SANS SERIF	[PORTUESL]
73	PENPALFONT		CARATTERI SCRITTURA BAMBINI (GUSTALI MENTRE SCRIVI!)	[OGDEN]
34	PINBALL		16 LINEE	[FISCHER]
34	PINOCCHIO		8 LINEE	[FISCHER]
135	PKFONTS	2.5	24 PK CARATTERI(15 A 150 PL) - USA TEXF PER CONVERTIRE	[OZER]
34	PUDGY		8 LINEE	[FISCHER]
41	QUARZ		CARATTERI 6X5 - SANS SERIF - DOPPIA AMPIEZZA IN VERTICAL	
115	ROBOTRON		6X8 LINEE - STILE COMPUTER -	
34	ROMAN		8 E 16 LINEE	[FISCHER]
34	SCRIPT		16 LINEE	[FISCHER]
50	SGTOPAZ		CARATTERI TOPAZ MODIFICATI (SPECIALE VT102 GRAFICO)	[WARKER]
34	SHADOW		8 LINEE	[FISCHER]
71	SHALT18		CARATTERI CONTENENTI ELEMENTI DI CIRCUITI ELETTRONICI	[KITTEL]
105	SHARP		8 LINEE - SIMILI A CARATTERI CLEAN	[DAVIES]
34	SKINNY		8 LINEE	[FISCHER]
34	SLANT		8 LINEE	[FISCHER]
34	SLOPE		16 LINEE	[FISCHER]
81	STFONT		8 LINEE - LUNGHEZZA FISSA - CARATTERI STILE ATARIST	
34	STOP		8 LINEE	[FISCHER]
41	TERMINAL		CARATTERI A 7 PUNTI	
98	THAI		22 LINEE - SET DI CARATTERI THAI	
34	TYPEWRITER		16 LINEE	[FISHER]

FONT UTILITY

60	BLITZFONTS		AUMENTA LA VELOCITA' DEL TESTO DI 6 VOLTE(UTENTE TRASP.)	[HAUGEN]
60	FONTEDITOR	1.1	PERMETTE DI EDITARE/CREARE CARATTERI	[ROBINSON]
138	MACFONT		CONVERTE CARATTERI PER MAC IN CARATTERI PER AMIGA	[O'NEILL/ MARIANI]
75	SETFONT	2.0 *	CAMBIA I CARATTERI IN CLI/WORKBENCH	[HAYNIE]
135	TEXF	2.5	CONVERTE CARATTERI PK IN CARATTERI AMIGA	[OZER]

GESTISCONO MUSICA

126	BMP		BACKGROUND MUSIC PLAYER (ESEMPIO DI MUSICA IFF)	[WHITE]
58	SMOSPLAYER	3.5	SUONATORE PER SMUS IFF MUSIC FILES (SECONDA VERSIONE)	[HODGSON]

GRAFICA

47	3D-ARM	*	ANIMAZIONE DELLE BRACCIA DI UN ROBOT IN 3D CHE SEGNANO	[LAUGHLIN]
99	A-RENDER	.3	PROGRAMMA PER RAY TRACING (CON ESEMPI)	[REED]
12	AMIGA3D		SEGNI DI ROTAZIONE 3D PER AMIGA -PUNTO DI VISTA MOVIBILE	[WHITEBOOK]
14	AMIGA3D	1.1 *	SEGNI DI ROTAZIONE 3D PER AMIGA - PUNTO DI VEDUTA MOVIB.	[WHITEBOOK]
12	ARROW3D		ROTAZIONE DI UNA FRECCIA IN 3D DISEGNATA PER LINEE	[BEATS]
1	BALLS	*	ESEMPIO DI MOTO PERPETUO	[KIVOLOWITZ]
49	BCS	1.1 *	SET DI COSTRUZIONE BONSAI- CRESCITA DI ALBERI RICORSIVA	[GINTS]
67	BOUNCER	*	PALLE RIMBALZANTI CON OMBRE RELATIVE IN 3D	[BRYAN]
71	CALEINDOSCOPE	*	CALEINDOSCOPIO COLORATO (AMIGABASIC)	[KITTEL]
1	COLORFUL	*	DIMOSTRAZIONE DEL MODO HOLD AND MODIFY	[PECK/ PARISEAU]
38	CSQUARED	*	ALGORITMO PER CERCHI REGOLARI	[DUPREE]
49	CYCLOIDS	1.1 *	PROGRAMMA PER DISEGNARE TIPI DI SPIRALI	[GINTZ]
14	DIMENSIONS	*	RUOTA UNA FRECCIA DISEGNATA CON LINEE	[HOOK]

1	HALFBRITE		DIMOSTRAZIONE DEL MODO EXTRA-HALF-BRITE	[HOOK]
118	HAMMMM	*	MOVIMENTO LINEE/AREE CON RELATIVI SUONI	[BURK]
27	HYPOCYCLOIDS	1.0	PROGRAMMA PER DISEGNARE TIPI DI SPIRALI	[GINTZ]
67	MANDALA		DISEGNO DI UN CALEIDOSCOPIO (CON MUSICA CASUALE)	
18	MULTIDIM	0.0 *	ROTAZIONE 2-6 CUBO DIMENSIONALE	[FRENCH]
15	POLYFRACTALS	*	DISEGNA IMMAGINI FRATTALI (ABASIC)	[ADDISON]
66	RAYTRACER		SEMPLICE PROGRAMMA DI RAY-TRACE	
67	RTCUBES		ROTAZIONE NELLO SPAZIO DI 16 CUBI IN 3D	[RUSSELL]
97	SHM	1.42 *	TRACCIA IL PERCORSO DI DUE PENDOLI INTERATTIVI	[EDDIS]
9	SKEWB	*	SEMPLICE ANIMAZIONE DEL CUBO DI RUBIK	[BRAND]
89	SNAKE	*	DEMO DI MOVIMENTI DI LINEE-CURVE E BAZIER SPLINES	[BJORKE]
32	SPIN3	*	CUBI IN ROTAZIONE - STILE OP ART	[PETERSON]
97	SPLINES		VARIE CURVE GENERABILI CON TECNICHE DI RENDERING	[TARAN]
31	TREE	1.00 *	DISEGNA ALBERI RICORSIVI	[FRENCH (BALTHROP)]
55	VSPRITES	*	DEMO USO DI VSPRITES	[COTTON]
61	VSPRITES	*	DEMO USO DI VSPRITES	[PECK]
118	WIREDEMO	*	DISEGNA PER LINEE ED ANIMA IL LOGO AMIGA	[DILLON]
89	ATV	*	GENERA BUONI SCENARI RANDOM	[HULL (GRAY)]
16	BMPRINTC.C	*	CONVERTE ILBM FILE IN ASCII PER PROGRAMMI C	[MORRISON/ SHAW]
64	BMPRINTC.C	*	CONVERTE ILBM FILE IN ASCII PER PROGRAMMI C	[MORRISON/ SHAW]
101	CIRPLANE	1.00 *	GENERATORE DI PIANI CIRCOLARI PÉR VIDEOSCAPE 3D	[FLORYAN]
87	CLAZ	2.0 *	CONVERTE FIGURE IFF(INC HAM) PER POSTSCRIPT	[LUDTKE]
126	COLOUR	*	MANIPOLA E SALVA SETS COLORATI DI SCHERMI CHIAMATI	[RUSSELL]
137	CT		IMMAGINI DI PROCESSO DI CATSCAN CERVELLO/CUORE IMMAGINI	[HARMAN]
85	FILTER	1.0	PROGRAMMA PER FILTRARE IMMAGINI IFF	[VERMEULEN]
71	FPIC		PERMETTE PROCESSI DI IMMAGINI SU SCHERMO	[BUSH]
52	FRACTAL	*	PRODUCE RETTANGOLI DI TERRENO FRATTALI CASUALI	[OLSEN]
14	GI	*	CONVERTE PENNELLI DEL DPAINT PER FILE SORGENTI C	[FARREN]
96	LINEDRAWER	1.0 *	FA LINEE DISEGNANDOLE DA COMANDI DI FILE TESTO	[OLSEN]
32	MACVIEW	1.01	VISUALIZZA FILES DI FIGURE MAC	[EVERDEN]
35	MACVIEW	1.01	VISUALIZZA FIGURE MAC (INC DELLE PIC)	[EVERDEN]
71	ROT	0.5	GENERE E VISUALIZZA OGGETTI IN 3D	[FRENCH]
87	SC	*	GENERA BUONI SCENARI CASUALI	[GRAY]
40	SPRITEED		EDITOR DI SPRITE	[LAMB]
61	TERRAIN	*	GENERA SCENARI FRATTALI	[GRAY]
94	TERRAIN3D	*	GENERA SCENARI FRATTALI	[HULL (GRAY)]

GRAFICI MATEMATICI

71	3D-NPLOT	*	PLOTTER GRAFICO IN 3D (AMIGABASIC)	[KITTEL (FASERINC)]
13	3DSOLIDS.BAS	*	DISEGNA OGGETTI/LINEE IN 3D DA DATI I/P (ABASIC)	[HUDSON/GROK]
13	ALG1.BAS	*	DISEGNA GRAFICI DI SENO, COSENO ECC.	[ARSDALE VAN]
13	ALGEBRA	*	DISEGNA GRAFICI DI SENO, COSENO ECC.	[ARSDALE VAN]
75	BCUBIC	*	ESPERIMENTO CON CUBI BAZIER (3D) SUPERFICI CURVE	[DILLON]
75	BEZIER	*	ESPERIMENTO CON CURVE BEZIER	[DILLON]
75	BSPLINES	*	ESPERIMENTO CON BSPLINES	[DILLON]
105	CURVES	*	SIGNORINA "CURVY" DA LEASTSQUARE (AMIGABASIC)	[TREPAL]
121	DATAPLOT	3.0 *	PLOTTING GRAFICO + PRECEDENTI RETTANGOLI (AMIGABASIC)	[HOLT]
58	EGRAPH		LEGGE VALORI XY DA UN FILE E DISEGNA IL FORMATO GRAFICO	[TURNER]
97	GRAPHIT	II *	DISEGNA SEMPLICI FUNZIONI IN 2 O 3 DIMENSIONI	[FISHMAN]
105	LEASTSQUARE	*	ADATTA CURVE USANDO I PRECEDENTI RETTANGOLI (AMIGABASIC)	[TREPAL]
56	MP	*	DISEGNA GRAFICI DA I DATI DI UN FILE	[GINTZ]
49	PLOT	*	COMPUTA E DISEGNA FUNZIONI 3D	[GINTZ]
121	PLOT	*	GRAFICI 3D - SALVA IN IFF (AMIGABASIC)	[TREPAL]
56	TDP	*	DISEGNA GRAFICI 3D DA I DSATI DI UN FILE	[MOONEY]

HARDWARE

66	AMSCSI		DOCUM. PER IL PROGETTO PER COSTRUIRE UN'INTERFACCIA SCSI	[FRANTZ]
65	JTIME		ISTRUZIONI E DIAG.PER LA COSTRUZIONE DEL CLOCK ESTERNO	[KERYAN]
27	MEMEXPANSION	*	SCHEMA ECC,PER COSTRUIRE 1 MEGA RAM + CLOCK	[FELLINGER]

IFF

16	IFF DISK	1	COMPLETO SVILUPPATORE DI IFF DISK	[COMMODORE-AMIGA]
64	IFF DISK	2	COMPLETO SVILUPPATORE DI IFF DISK	[COMMODORE-

AMIGA]

IFF DOC

16	8SVX	7-2-85	DESCRIZIONE DI EA IFF FILES DI ESEMPI DI VOCE	[MORRISON]
64	8SVX	7-2-85	DESCRIZIONE DI EA IFF FILES DI ESEMPI SI VOCE	[MORRISON]
16	BACKGRND.DOC	86/01	BACKGROUND PER IFF ESEMPI DI PROGRAMMI SU FISH.16	[MORRISON]
64	BACKGRND.DOC	86/01	BACKGROUND PER IFF ESEMPI DI PROGRAMMI SU FISH.64	[MORRISON]
64	COMPILINGIFF		CONSIGLI PER COMPILARE GLI ESEMPI IFF	[SCHEPPNER]
16	EA.IFF.85		DESCRIZIONE DEL FORMATO IFF DELL'ELECTRONIC ARTS	[MORRISON]
64	EA.IFF.85		DESCRIZIONE DEL FORMATO IFF DELL'ELECTRONIC ARTS	[MORRISON]
16	FTXT	15-1185	DESCRIZIONE DEL FORMATO IFF TEXT DELL'ELECTRONIC ARTS	[MORRISON/ BURNS]
64	FTXT	15-1185	DESCRIZIONE DEL FORMATO IFF TEXT DELL'ELECTRONIC ARTS	[MORRISON/ BURNS]
16	ILBM	17-1-86	DESCRIZIONE DEL FORMATO IFF ILBM DELL'ELECTRONIC ARTS	[MORRISON]
64	ILBM	17-1-86	DESCRIZIONE DEL FORMATO IFF ILBM DELL'ELECTRONIC ARTS	[MORRISON]
44	NEWIFF	*	NUOVA VERSIONE DI ALCUNI FILE DEL DISCO 16	[SCHEPPNER]
64	README.DOC		VISUALIZZAZIONE DI ESEMPI DI FILE IFF SU FISH.64	[ELECTRONIC ARTS]
16	README.DOC	86/02	VISUALIZZAZIONE DI ESEMPI DI FILE IFF SU FISH.16	[ELECTRONIC ARTS]
64	SMUS		DESCRIZIONE DEL FORMATO IFF MUSIC DELL'ELECTRONIC ARTS	[MORRISON]
16	SMUS	5-2-86	DESCRIZIONE DEL FORMATO IFF MUSIC DELL'ELECTRONIC ARTS	[MORRISON]

LETTORI DI FILE

36	PRINTEXT	1.2	VISUALIZZA FILE TESTO - CON MOLTE CARATTERISTICHE	[KAHANE]
85	VMORE	1.00 *	LETTORE DI TESTI (COME MORE) PER PIU' FILE	[VERMEULEN]
85	VNEWS	*	LEGGE TUTTI I FILES DI UNA SPECIFICATA DIRECTORY	[VERMEULEN]
107	VNEWS	1.1 *	VISUALIZZA I CONTENUTI DEI FILES SELEZIONATI	[VERMEULEN]

LINGUAGGIO 6502

92	AS6502	*	ASSEMBLER PER 6502	[SWANK (ORNUM)]

LINGUAGGIO ICON

81	ICON	6.0	LINGUAGGIO DI PROGRAMMAZIONE ICON DA UNIV.DELL'ARIZONA	

LINGUAGGIO POSTSCRIPT

101	PSINTRP	*	INTERPRETE POSTSCRIPT (BASATO SOLO SULLO SCHERMO)	[LEE]

LINGUAGGIO SMALLTALK

37	SMALLTALK	1 *	SISTEMA SMALLTALK PORTATO SU AMIGA	[BUDD/ KINNERSLEY]

MANDELBROT

80	IMANDELVROOM		GENERA PUNTI VELOCEMENTE	[CLAGUE/HULL]
90	IMANDELVROOM		GENERATORE MANDELBROT CON ANELLO RIVELATORE	[HULL (CLAUGUE)]
4	MANDEL	1.0 *	UN PROGRAMMA PER MANDELBROT	[FRENCH/MICAL]
111	MANDEL	1.0 *	SET DI COSTRUZIONE MANDELBROT	[SEIBERT]
13	MANDEL.BAS	*	DISEGNA MANDELBROT IN UNA PICCOLA AREA (ABASIC)	[SEIBERT]
13	MANDELBROT	*	PROGRAMMA MANDELBROT SET (ABASIC)	
5	MANDELBROT	2.01 *	GENERATORE MANDELBROT-CON SUPPORTO IFF	[FRANCH/MICAL]
31	MANDELBROT	3.00 *	PROGRAMMA DI CREAZIONE MANDELBROT	[FRENCH]
20	MANDELBROTS		SET DI FIGURE MANDELBROT	
75	MANDELVROOM	1.50 *	GENERA PUNTI VELOCEMENTE	[CLAGUE]
95	MANDFXP-D3	3.0	DEMO DI UN PROGRAMMA MANDELBROT	[CYGUNSSOFT]
21	MSE		PACCHETTO ESPLORATORE MANDELBROT SET	[WILCOX]
130	QMAN	*	GENERATORE MANDELBROT VELOCE	[BONNER]

MATEMATICA

71	AIRFOIL	*	VISUALIZZA SUPERFICI AEREODINAMICE DATE DAI PARAMETRI	[LEIGHTON]
74	UNITS	*	CONVERTE SET DI UNITA' E GENERA CARTE	[SIMPSON]

MIDI

82	DX-SYNTH		DX-7 LIMITATORE DI VOCE (NUOVA VERSIONE DI VOICEFILER)	[DECKARD]
101	MIDI	*	LIBRERIA DI ROUTINE PER MULTITASKING CON MIDI	[BARTON]
54	MIDITOOLS	*	SET DI PROGRAMMI PER REGISTRARE/ASCOLTARE COM MIDI	[CASSIRER]
82	PANL	1.2	PANNELLO MIDI UNIVERSALE	[WEINBACH]
38	VOICEFILER	*	PERMETTE A VOCI DX DI ESSERE SALVATE SU DISCO	[DECKARD]

MOSTRANO FIGURE

58	BIGVIEW	*	VISUALIZZATORE DI FIGURE PER OGNI FORMATO DI FIGURE	[HODGSON]
64	CYCLES		MOSTRA FIGURE ILBM - CON COLORI CICLICI	
17	DIGIVIEWER		VISUALIZZATORE DI FIGURE PER DIGIWIEW MODO HAM PICS	
64	DISPLAY		SEMPLICE VISUALIZZATORE DI FIGURE ILBM	
39	DISPLAY	1.0 *	VISUALIZZATORE PER COLLEZIONE DI FIGURE RAY TRACED(LENTO	[WECKER]
73	DISSOLVE	*	FIGURE IFF DISSOLTE SULLO SCHERMO(PIXEL DOPO PIXEL)	[ROBERTSON]
11	DPSLIDE	1.0	VISUALIZZA SEQUENZE DI IMMAGINI IFF CHE SCORRONO	[BIONDO]
30	DPSLIDE	1.0	SCORRITORE DI SEQUENZE	[BIONDO]
27	LOADILBM	*	CARICA/VISUALIZZA FIGURE IFF ILBM - SALVO COME ACBM	[SHEPPNER]
35	PLOP	*	SEMPLICE,NIENTE INTRECCI,VISUALIZZATORE DI FIGURE	[KENT]
16	SEEILBM	*	VISUALIZZA SINGOLE FIG ILBM FINCHE' LA WINDOW E' CHIUSA	[SCHEPPNER]
64	SEEILBM		VISUALIZZA SINGOLE FIG ILBM FINCHE' LA WINDOW E' CHIUSA	[SCHEPPNER]
60	SHOW	2.1	VELOCE SLIDESHOW (TUTTA RES + HAM)	[RACHMAT]
32	SHOWHAM		VISUALIZZA IMMAGINI HAM(IMMAGAZINA E MODIFICATE NON IFF)	
10	SHOWILBM	*	MOSTRA SPECIFICATE FIGURA/E ILBM/IFF	[MORRISON/ SHAW/HAYES]
11	SHOWILBM		MOSTRA SPECIFICATE FIGURA/E ILBM/IFF	[MORRISON/ SHAW/HAYES]
16	SHOWILBM	*	MOSTRA SPECIFICATE FIGURA/E ILBM/IFF	[MORRISON/ SHAW]
30	SHOWILBM		MOSTRA SPECIFICATE FIGURA/E ILBM/IFF	
64	SHOWILBM	*	MOSTRA SPECIFICATE FIGURA/E ILBM/IFF	[MORRISON/ SHAW]
58	VIEW	1.8 *	VISUALIZZATORE ILBM MOLTO PICCOLO	[HODGSON]
44	VIEWILBM	*	LEGGE NORM/HAM ILBM FILES (BASATO SU SHOWILBM)	[SHEPPNER]
116	VILBM		VISUALIZZADELLE IMMAGINI IFF O FILE TESTO	[GRAHAM]
87	YAIFFR	*	LETTORE DI FILE IFF FIGURE (TUTTI I MODI)	[SCHWAB]

MUSICA

126	MOUNTAINKING		GRIEG'S MOUNTAIN KING - ESEMPIO SI MUSICA (BMP)
125	YTV		YOU'RE THE VOICE - ESEMPIO DI MUSICA (SONIX)

PROGRAMMANDO LA GRAFICA

33	BIGMAP	*	MOSTRA L'USO DI SCROLLVPORT	[SCHWAB]
33	DBUG.GELS	*	DEMO DI ANIMAZIONE USANDO BOBS E VSPRITES	[LUCAS]
41	DUALPLAYFIELD	*	MOSTRA L'USO DI UN CAMPO DI GIOCO DUALE	[MACKRANZ]
15	FISH	*	PESCI CHE NUOTANO-(ANIMOB E DOPPIO-BUFFERED)	[WAGNER]
5	LAYERS	*	ESEMPIO DI USO DI LIBRERIE LAYERS	[PECK]
27	LINESDEMO	*	SCROLL ATTORNO SUPERBITMAP CON GADGETS PROPORZIONALI	[WHITEBOOK]
130	PATEDIT	*	CREA PATTERNS PER CHIAMATE MACRO SETAFPT	[HYDE]
52	POLY	*	DEMO DI AREAMOVE,AREADRAW,AREAEND	[OLSEN]
5	REGION	*	MOSTRA LE REGIONI UTILIZZATE	[PECK]
5	SAMPLEFONT		PRODUCE CARATTERI MAZZE,CUORI,SPADE E DIAMANTI	
35	SCROLLPF	*	ESEMPIO DI CAMPO DI GIOCO DUALE	[SCHEPPNER]
5	SINGLEPLAYFIELD	*	VISUALIZZA UN SINGOLO CAMPO DI GIOCO 300X200 2-BITPLANE	[PECK]
35	SPRITEMAKER		DISEGNA SPRITES E LI SALVA COME STRUTTUTE DATI C	[LARSON]
28	SUPERBITMAP	*	MOSTRA L'USO DI SCROLLLAYER E SUPERBITMAP	[SCHEPPNER/ LINDSAY]
87	WBDUALPF	*	ESEMPIO DI SCHERMO USANDO UN CAMPO DI GIOCO DUALE	[MACKRANZ]

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 ☐ #3 ☐ #4 ☐ #5 ☐ #6	Lit 15.000

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

Cognome .....

Nome .....

Via .....

Cap. ..... Città .....

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

Firma .....

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

Linguaggio Assembly

Parte 2 - argomenti della linea di comando

di Jim Butterfield

Jim Butterfield non ha bisogno di essere presentato. Il suo nome costituisce un punto di riferimento fra gli entusiasti utenti Commodore di tutto il mondo. Ha cominciato a interessarsi di microcomputer ai tempi del KIM-1 da 1K. Jim possiede una conoscenza enciclopedica dei prodotti Commodore, come testimoniato da articoli, libri, lezioni e addirittura da programmi televisivi.

La volta scorsa abbiamo toccato molti argomenti. Questa volta intendo metterne in evidenza qualcuno e aggiungere un po' di prospettiva. Nella prima parte abbiamo dovuto galoppare velocemente per arrivare al punto di potere scrivere un programma funzionante.

Esplorare è divertente, ma ora è tempo di leggere questa lista, assicurandoci che la nostra base di conoscenze sia solidamente al suo posto.

1 - I registri *dati* e *indirizzi* del 68000 possono contenere ognuno 4 *byte* di informazione. Se preferite, potete chiamarli 32 *bit*, 2 *word* o 1 *long word*. Questo significa che possiamo metterci dei numeri abbastanza grossi, lì dentro, e che possiamo farci sopra delle operazioni aritmetiche.

2 - I registri dati possono essere usati in una delle tre seguenti maniere: come intere *long word*, tutti e quattro i byte; come *word*, i due byte inferiori; o come *byte*, il byte inferiore del registro. La parte del registro che non utilizziamo non verrà influenzata dalle nostre operazioni.

Le operazioni sui registri indirizzi coinvolgono l'intera *long word*. Questo vale anche se specifichiamo il modo .W (word); i due byte che modifichiamo verranno espansi per riempire l'intero registro indirizzi. Non possiamo usare il modo .B (byte) con un registro indirizzi.

3 - Il registro indirizzi A7 è lo *stack*; non modificatelo senza ragione. Il registro indirizzi A6 viene usato in Amiga per le chiamate alle librerie; lasciamolo ragionevolmente libero per questo scopo. I registri dati da A0 ad A5 inclusi e tutti i registri dati sono a nostra disposizione per essere utilizzati. Tenderemo a usare prima i registri con numerazione alta; questo perché le chiamate alle subroutine possono cancellare l'informazione contenuta nei registri a bassa numerazione, come il prossimo punto meglio specifica.

4 - Le subroutine di sistema (le chiamate alle librerie) fanno attenzione a non cambiare i contenuti dei registri indirizzi e dati numerati da 2 a 7; i valori in essi contenuti sono al sicuro. Ma i registri A0,A1,D0 e D1 possono essere modificati da queste subroutine; chiamando una subroutine perderemo ogni informazione contenuta nei suddetti registri.

5 - Il 68000 può raggiungere la memoria un byte alla volta (a qualsiasi indirizzo), oppure due byte (una *word*) alla volta (il byte inferiore a un indirizzo pari, il byte superiore all'indirizzo dispari immediatamente seguente). Un'istruzione può chiedere alla CPU di gestire quattro byte (una *long word*), nel qual caso verranno effettuati due cicli da due.

Potremmo, quindi, chiedere al processore di darci il contenuto della long word all'indirizzo 4, nel qual caso verrebbero letti i contenuti dei byte 4, 5, 6 e 7 e il risultato verrebbe messo nel registro specificato. Se tentassimo di effettuare una operazione su long word (o su word) all'indirizzo 5 comunque, il computer si rifiuterebbe e incontreremmo il guru.

6 - Abbiamo visto alcune istruzioni del 68000. Per muovere i dati abbiamo usato MOVE, che sposta l'informazione (byte, word o long word). LEA (Load Effective Address) sposta un indirizzo (come opposto allo spostare dati di MOVE) in un registro indirizzi.

Queste istruzioni hanno due *operandi*, un *da* e un *per*, o più formalmente, una *sorgente* e una *destinazione* (source and destination). MOVE specifica normalmente la lunghezza dei dati su cui opera, precisamente .L per long word, .W per word e .B per byte. Pertanto, MOVE.L 4,A6 sposta i quattro byte dagli indirizzi 4, 5, 6 e 7 nel registro indirizzi A6. LEA sposta sempre un indirizzo completo, che è composto da una long word.

BSR (Branch Subroutine) chiama una subroutine che sicuramente risiede nelle vicinanze (entro un raggio di 32000 byte circa). Se la subroutine si trova veramente vicino, entro un raggio di 120 byte circa, potremo risparmiare un po' di memoria specificando BSR.S (salto alla subroutine, corto (short)). In

contrasto, JSR (Jump Subroutine) può chiamare una subroutine in qualsiasi zona della memoria, magari una che è troppo lontana per essere raggiunta con BSR.

BSR e JSR hanno un operando: la locazione della subroutine che vogliamo chiamare.

RTS ritorna da una subroutine al punto in cui era stata chiamata (termina l'esecuzione della subroutine, restituendo il controllo all'istruzione successiva). L'intero programma che noi scriviamo è una subroutine chiamata dal sistema operativo. Quando è finito, dovrebbe restituire il controllo al sistema con RTS. l'istruzione RTS (ReTurn from Subroutine) non ha operandi; è completa in sé stessa.

7 - Abbiamo visto alcuni *modi di indirizzamento*. Facciamo un sommario di quelli più visibili.

Diretto di registro, significa il contenuto stesso del registro. Quindi, MOVE.L A0,D0 significa 'sposta il contenuto di A0 nel registro D0'.

Indiretto da registro, significa che un registro indirizzi contiene l'indirizzo del dato. Vengono usate le parentesi per indicare un indirizzamento indiretto; solo i registri indirizzo possono essere usati in questo modo. Quindi, MOVE.L (A1),D0 significa: "A1 contiene un indirizzo; sposta il contenuto di quell'indirizzo nel registro D0". In questo esempio, l'indirizzo contenuto in A1 deve essere un numero pari, altrimenti...

Indiretto da registro con spiazzamento, ancora una volta richiede un indirizzamento indiretto, ma esegue alcune operazioni aritmetiche supplementari, calcolando l'offset (o spiazzamento) per ricavare l'indirizzo usando la costante numerica che viene fornita. Quindi, JSR -$228(A6) significa: "prendi l'indirizzo da A6, sottraigli il numero esadecimale 228 e chiama la subroutine all'indirizzo così risultante".

Assoluto, specifica un indirizzo della memoria di Amiga. Usiamo raramente indirizzi fissi, eccetto che per l'indirizzo 4, il quale contiene un puntatore alla nostra libreria Exec. Anche l'input/output e i chip di sistema non dovrebbero essere normalmente consultati direttamente dal programmatore. Bisognerebbe chiamare, invece, una funzione di libreria che leggerà i dati per nostro conto.

MOVE.L 4,A6 trasferisce il contenuto dell'indirizzo 4 (e del 5, 6 e 7) nel registro A6. Notate che questo è diverso da LEA 4,A6 che sposterebbe 4, l'indirizzo, in A6.

Soprendentemente, possiamo usare l'indirizzamento assoluto per riferirci a qualcosa all'interno del nostro stesso programma. A prima vista questo potrebbe sembrare bizzarro, dal momento che non sappiamo dove si troverà il nostro programma quando verrà caricato in memoria. Comunque sia, tutto funziona bene; l'assembler e la funzione che si occupa di caricare i programmi nella memoria di Amiga (loader) lavorano di comune accordo per sistemare gli indirizzi in modo che siano corretti e che rispecchino la realtà, ogni volta che il nostro programma verrà eseguito.

Pertanto, se abbiamo un byte di informazione memorizzata all'indirizzo *VALUE* all'interno del nostro programma, potremo scrivere MOVE.B VALUE,D5 per trasferire il contenuto di VALUE nel byte inferiore del registro D5. Quando sia possibile, comunque, dovremmo cercare di evitare questo tipo di indirizzamento a favore dell'indirizzamento relativo a PC, che adesso vedremo.

Indiretto da Program Counter con spiazzamento, chiamato brevemente *relativo*. Dice al processore di guardare avanti o indietro dalla sua attuale posizione di lavoro (nel raggio di circa 32000 byte in entrambe le direzioni) per trovare la locazione interessata. Useremo spesso questo indirizzamento per riferirci a dati contenuti all'interno del blocco corrente del nostro programma.

Non si può usare sempre questo modo di indirizzamento, ma quando è possibile si risparmiano memoria e tempi morti. Dal momento che l'offset può essere espresso in due byte e che un indirizzo completo richiederebbe una long word di quattro byte, possiamo risparmiare spazio e tempo. Questo tipo di indirizzamento è valido sempre, non importa in quale zona venga caricato il nostro programma. In contrasto, un indirizzamento assoluto che si riferisce a una locazione all'interno del nostro programma dovrà sempre essere sistemato a seconda di dove va a finire il programma.

Il nostro esempio precedente di MOVE.B VALUE,D5 può essere vantaggiosamente trasformato in MOVE.B VALUE(PC),D5. Il codice risulta più corto e non richiede di essere rilocato durante il caricamento.

Il modo (PC), comunque, può essere usato solo come parte dell'operando sorgente, il che vuole dire, solo come primo indirizzo. Se il nostro esempio avesse voluto spostare il dato all'incontrario, avremmo dovuto scrivere MOVE.B D5,VALUE poiché (PC) non sarebbe stato disponibile.

Immediato, fornisce un valore reale da utilizzare. Si distingue per l'uso del carattere # (chiamato, a volte, cancelletto). Quindi, MOVE.L #0,A4 mette il valore zero nel registro A4.

Come succede con l'indirizzamento (PC), quello immediato può essere usato solo nel primo operando. Così, se vogliamo comparare il contenuto di D4 con il numero cinque, dobbiamo scrivere CMP.L #5,D4 e non viceversa.

Esistono altri modi di indirizzamento, ma li vedremo man mano che si presentano.

8 - C'è bisogno di un *codice di inizializzazione* (startup), una serie di istruzioni che fanno un lavoro di preparazione. Questo comprende cose come aprire le librerie di sistema prima di affrontare il compito pricipale e chiuderle quando sia finito. Magari sarebbe più accurato chiamarlo codice di startup-e-closedown (di inizializzazione e chiusura).

Il codice di startup che abbiamo scritto l'ultima volta presupponeva tre cose. Primo, che il programma sarebbe stato chiamato solo da CLI. Questo ci ha risparmiato una serie di

complesse operazioni. Tra le altre cose, potevamo contare sul fatto che ci fosse una finestra alla quale mandare il nostro output.

Secondariamente, assumeva che l'unica libreria di cui avremmo avuto bisogno era quella del DOS. Per un principiante questo è comodo; il DOS rappresenta il percorso più semplice per un output sullo schermo, sul disco, sulla stampante o su altro. Il DOS comunque non è molto utile per disegnare immagini o suonare musica; ma ci sarà tempo per queste cose più avanti.

La nostra terza assunzione, una che cambieremo oggi, consisteva nel fatto che il comando CLI era formato dal solo nome del programma. Questo significa che il programma verrebbe invocato digitando *TEST*. Se scegliessimo di scrivere *TEST THIS PROGRAM*, la parte rimanente della linea, *THIS PROGRAM*, verrebbe ignorata. In questa sessione inizieremo a vedere come si fa a leggere la 'coda' di un comando, come è chiamata.

9 - Un'osservazione: nello scrivere un programma interamente auto-contenuto, abbiamo fatto qualcosa che è abbastanza inusuale nel mondo Amiga. La maggior parte dei programmi, nella loro zona iniziale, richiamano definizioni di dati supplementari da file chiamati *include*. Molti programmi usano le *macro* per generare codice prefabbricato che non ha bisogno di essere ripetuto. È cosa comune per i programmi Amiga il richiamare codice di startup già pronto. Noi lo faremo più avanti, dopo avere scritto sezioni di startup più complete. Molti programmatori usano la tecnica di scrivere e assemblare intere sezioni di codice; più tardi, il linker combinerà questo codice con altri segmenti per creare un programma completo e funzionante.

Infine, è uso pressoché universale quello di chiedere al linker di sostituire gli indirizzi delle librerie al nostro posto; è veramente troppo pretendere di ricordare che -$228 è l'entrata di *OpenLibrary*. Faremo solo un accenno a quest'area al momento. Ogni diverso sistema di assembler/linker ha regole differenti. Noi manterremo il controllo di tutto quello che è possibile, in modo che ci sia la minima dipendenza da quelle che sono le caratteristiche particolari di un sistema di sviluppo.

Il prossimo progetto: ECHO al contrario

Sappiamo che il comando ECHO del CLI ripete quello che trova nella coda della linea di comando; questo, perlomeno, è il suo compito principale. Il nostro progetto consiste nello scrivere un comando ECHO che ripeta le informazioni all'incontrario. Non sarà una cosa molto utile, ma avremo modo di imparare una serie di cose lungo la strada. Chiameremo questo programma OHCE... ECHO pronunciato alla rovescia.

Apriamo un nuovo file chiamato OHCE.ASM sul nostro editor favorito e mettiamoci al lavoro. Alcune linee di commento e poi ci dedicheremo alle definizioni delle librerie di sistema. Esse saranno le stesse che abbiamo visto nel nostro precedente programma, ma questa volta, lasceremo che il sistema si occupi di trovarne il valore.

```
; Programma ECHO al contrario
; Utilizzabile solo da CLI o Startup-Sequence
; Definizioni per la libreria dell'Exec

XREF _LVOOpenLibrary
XREF _LVOCloseLibrary

; LVO significa Library Vector Offset
```

Il comando XREF dice all'assembler: "Lascia uno spazio ogni volta che vedi queste etichette, insieme a una nota per il linker che deve riempire questi spazi con il valore individuato dall'etichetta stessa". L'assembler allora continua il suo lavoro; anche se non conosce il valore di, mettiamo, _LVOOpenLibrary, non segnala alcun errore.

Il linker, a sua volta, legge il file oggetto e capisce di dover trovare queste etichette in qualche altro file: in un altro file oggetto o in un file *libreria*. Quando trova l'etichetta, ne riempe gli spazi con il valore appropriato. Risultato: alla fine otteniamo un programma completo ed eseguibile.

Non possiamo usare XREF con l'assembler *AssemPro*, il quale non supporta la fase di linking. Dobbiamo, invece, sostituire il comando con:

```
INCLUDE INCLUDE/EXEC.OFFSETS
```

Questa operazione inserirà automaticamente una serie di definizioni EQU per _LVOOpenLibrary, _LVOCloseLibrary e anche di tutte le altre funzioni dell'Exec che non usiamo in questo programma. Dovremo ripetere quest'operazione quando arriveremo alle definizioni della libreria del DOS.

Acchiappiamo la coda

Il nostro codice di startup precedente iniziava aprendo la libreria del DOS. Adesso abbiamo qualcosa di nuovo da fare prima di occuparci di quello. Le informazioni che riguardano la coda della linea di comando ci vengono fornite dal loader di Amiga: A0 contiene l'indirizzo di questa stringa di testo e D0 ne contiene la lunghezza.

I registri A0 e D0 saranno alterati nei loro contenuti quando chiamiamo una subroutine di sistema, quindi ci conviene parcheggiare i loro valori da qualche parte al sicuro. Qualsiasi registro con numerazione 2 o superiore andrà bene a questo scopo, quindi codificheremo:

```
MOVE.L  A0,A5
MOVE.L  D0,D5
```

Avremmo potuto scegliere qualsiasi altro registro conveniente per immagazzinare questi valori, tenendo a mente che A6 e A7 hanno altri compiti da svolgere. Adesso possiamo tranquillamente aprire la libreria del DOS.

```
MOVE.L  $4,A6           ; base dell'Exec
LEA     DosName(PC),A1  ; puntatore nome DOS
MOVEQ   #0,D0           ; qualsiasi versione
```

```
    JSR     _LVOOpenLibrary(A6)
    MOVE.L  D0,A6           ; puntatore base DOS
    BEQ.S   Exit            ; zero, esci
    BSR.S   Main            ; fai il tuo lavoro
    MOVE.L  A6,A1           ; puntatore base DOS
    MOVE.L  $4,A6           ; puntatore base Exec
    JSR     _LVOCloseLibrary(A6)


Exit:
    RTS
```

Questo completa il nostro codice di startup. È pressapoco lo stesso che abbiamo usato la volta scorsa, eccetto che per le istruzioni XREF e per il salvataggio dei nostri registri 'di coda'.

Avanti con la parte principale del programma, adesso. Come prima, trasformeremo le definizioni della libreria in istruzioni XREF e quindi chiameremo l'apposita funzione per ottenere il file handle del canale di uscita.

```
Main:
    ; parte principale
    ; definizioni per la libreria del DOS

    XREF    _LVOOutput
    XREF    _LVOWrite
    ;
```

Ancora una volta, non possiamo usare XREF con AssemPro, quindi scriveremo:

```
INCLUDE INCLUDES/DOS.OFFSETS
```

Questo comando inserirà le definizioni EQU per _LVOOutput e _LVOWrite oltre a parecchie altre. I file .OFFSET sono in formato testo; potete esaminarli se volete.

```
    JSR     _LVOOutput(A6)  ; prendi output handle
    MOVE.L  D0,D4           ; metti handle in D4
```

Avremo bisogno di utilizzare l'handle più di una volta... in D4 sarà al sicuro dalla cancellazione effettuata dalle subroutine.

Ricordate la coda della linea di comando? Il nostro compito consiste nello stamparla in senso inverso. Abbiamo la locazione di questi caratteri nel registro indirizzi A5 e il numero di questi caratteri nel registro D5. I caratteri sono byte, naturalmente, quindi useremo la funzione *Write* per stamparli. Non esiste alcun problema a stampare i caratteri uno alla volta e questo è esattamente quello che faremo, usando un loop. Programmi più sofisticati potrebbero copiare il messaggio in ordine inverso in una zona di lavoro della memoria e quindi stampare l'intera stringa in un colpo solo.

In ogni caso, la coda di una linea di comando *non* è terminata da un byte a zero. Normalmente troveremo lì un carattere di newline (linefeed). Se dobbiamo trovare la fine della coda, è generalmente più sicuro cercare un qualsiasi carattere non stampabile (qualsiasi valore inferiore a 32 decimale). Ma in questo caso non abbiamo bisogno di fare ciò, poiché conosciamo già la lunghezza della stringa: è contenuta in D5.

Penso che possiamo ragionevolmente ipotizzare che la stringa abbia una lunghezza inferiore a 65535 caratteri. Questo significa che non dobbiamo trattare D5 come long word; operando su word saremo sicuri di avere l'intero valore e risparmieremo anche una piccola (trascurabile) quantità di tempo di esecuzione.

```
Loop:
    SUBQ.W  #1,D5
    BMI.S   Finish
```

L'istruzione SUB (subtract, sottrai) sottrae 1 da D5. In questo caso abbiamo a che fare con un valore piccolo, cosa che ci permette di usare il comando SUBQ (subtract quick, sottrai veloce). Se il risultato va sotto zero verrà effettuato il Branch Minus (BMI) che salterà all'uscita del loop. In caso contrario avremo qualcosa da stampare:

```
    LEA     0(A5,D5.W),A0   ; indirizzo carattere
```

Aha! Un nuovo modo di indirizzamento! Questo viene chiamato *indiretto da registro con spiazzamento e indice*. Il nome incute rispetto, ma possiamo vedere in effetti cosa fa, semplicemente osservandolo attentamente. L'indirizzo in A5 viene sommato al valore di indice (lungo una word) contenuto in D5, poi al risultato viene aggiunto lo spiazzamento (zero). Durante il primo passaggio nel loop questa 'formula' ci fornisce l'indirizzo *dell'ultimo* carattere della coda. Man mano che attraversiamo ripetutamente il loop, decrementando D5, raggiungeremo i caratteri che lo precedono. Naturalmente *Write* ha bisogno che questo indirizzo gli venga fornito tramite il registro D2, quindi aggiungiamo un passaggio in più.

```
    MOVE.L  A0,D2           ; .. in D2
    MOVEQ   #1,D3           ; lunghezza carattere
    MOVE.L  D4,D1           ; file handle
    JSR     _LVOWrite(A6)   ; spedisci carattere
    BRA.S   Loop
```

Per essere precisi e ordinati, dobbiamo stampare un newline prima che il programma termini. Possiamo assumere che il registro D3 contenga ancora il valore 1?

```
Finish:
    LEA     NewLine,A0
    MOVE.L  A0,D2
    MOVE.L  D4,D1
    JSR     _LVOWrite(A6)
    RTS


DosName     dc.b 'dos.library',0
NewLine     dc.b $a


    end
```

L'intero lavoro

Ecco l'intero programma in un solo pezzo. Tenete bene a mente che le etichette (a volte chiamate simboli) devono essere posizionate completamente a sinistra affinché la maggior parte degli assembler le tratti correttamente. Alcuni assembler vogliono che anche le linee di commento inizino in corrispondenza al margine sinistro.

```
    ; Programma ECHO all'incontrario
    ; Utilizzabile solo da CLI o Startup-Sequence
    ; Definizioni per la libreria dell'Exec

    XREF _LVOOpenLibrary
    XREF _LVOCloseLibrary

    ; LVO significa Library Vector Offset

    MOVE.L  A0,A5
    MOVE.L  D0,D5
    MOVE.L  $4,A6              ; base dell'Exec
    LEA     DosName(PC),A1     ; puntatore nome DOS
    MOVEQ   #0,D0              ; qualsiasi versione
    JSR     _LVOOpenLibrary(A6)
    MOVE.L  D0,A6              ; puntatore base DOS
    BEQ.S   Exit               ; zero, esci
    BSR.S   Main               ; fai il tuo lavoro
    MOVE.L  A6,A1              ; puntatore base DOS
    MOVE.L  $4,A6              ; puntatore base Exec
    JSR     _LVOCloseLibrary(A6)

Exit:
    RTS

Main:
    ; parte principale
    ; definizioni per la libreria del DOS

    XREF    _LVOOutput
    XREF    _LVOWrite
    ;

    JSR     _LVOOutput(A6)     ; prendi output handle
    MOVE.L  D0,D4              ; metti handle in D4

Loop:
    SUBQ.W  #1,D5
    BMI.S   Finish

    LEA     0(A5,D5.W),A0      ; indirizzo carattere
    MOVE.L  A0,D2              ; .. in D2
    MOVEQ   #1,D3              ; lunghezza carattere
    MOVE.L  D4,D1              ; file handle
    JSR     _LVOWrite(A6)      ; spedisci carattere
    BRA.S   Loop

Finish:
    LEA     NewLine,A0
    MOVE.L  A0,D2
    MOVE.L  D4,D1
    JSR     _LVOWrite(A6)
    RTS

DosName     dc.b 'dos.library',0
NewLine     dc.b $a

    end
```

Assembliamo

Digitiamo tutto quanto nel nostro editor e salviamo su disco, usando il nome OHCE.ASM. Fate attenzione, ancora una volta, a maiuscole e minuscole nei nomi. Non abbiamo ancora bisogno di un printout... l'assembler generalmente lo fa per noi. Tenete a mente che se usate AssemPro, dovete cambiare le linee XREF con comandi INCLUDE e dovrete inoltre saltare il passo che utilizza il BLINK descritto più sotto.

Adesso facciamo partire l'assembler. In alcuni casi questo significa scegliere il comando Assemble da un menu, in altri significa invocare l'assembler da CLI. Dovremo scrivere:

```
(Metacomco):       ASSEM OHCE.ASM -o OHCE.OBJ
(Charlie Gibbs):   A68K OHCE.ASM OHCE.O
(Devpac2):         GENIM2 OHCE.ASM -L
```

Notate che specifichiamo i nomi del sorgente (.ASM) e dell'oggetto (.O o .OBJ); non abbiamo usato il percorso (path) completo per arrivare a questi file perché potreste volerli mettere in RAM:, piuttosto che in DF0: o in DF1: o addirittura nell'hard-disk. Per avere il file di listato, dovremo espandere leggermente le linee di comando:

```
(MCC):  ASSEM OHCE.ASM -o OHCE.OBJ -l OHCE.LST
(CG):   A68K OHCE.ASM OHCE.O OHCE.LST
(D2):   GENIM2 OHCE.ASM -L -P
```

I nomi dei file possono essere a vostro piacimento, ma i suffissi come .ASM (o .S), .OBJ (o .O) e .LST aiutano a capire subito cosa contengono. L'assembler C.A.P.E. 68K ha un menu chiamato *Options* che ci permette di scegliere i nomi per i file oggetto e listato; possiamo anche chiamare l'assembler da CLI se vogliamo. Per esempio:

```
(C.A.P.E. 68K):
  CAPE -a OHCE.ASM -o OHCE.OBJ -l OHCE.LST
```

L'assembler non ha trovato i valori dei simboli _LVO. Va tutto bene: lascerà un messaggio al linker.

Linkiamo

Il nostro programma adesso è contenuto ancora in un solo blocco; per un verso, infatti, non ha bisogno di essere linkato insieme ad altro. Dobbiamo, comunque, riempire i valori _LVO mancanti e quindi dobbiamo usare il linker per ottenere un file *eseguibile*, cioè, un programma che può essere lanciato.

Un assembler commerciale è generalmente fornito di una serie

di file *include*; la licenza per la riproduzione di questi file deve essere acquistata dalla Commodore e quindi non si troveranno mai insieme a un prodotto di pubblico dominio. Gli indirizzi _LVO di cui abbiamo bisogno, si trovano normalmente in un file chiamato *Amiga.lib*. Se non avete questo file, ne esiste uno equivalente e liberamente distribuibile chiamato *small.lib*, prodotto da Bryce Nesbitt. Potete trovarlo sui dischi di pubblico dominio o in una BBS. Avremo bisogno di ottenere le informazioni relative agli _LVO da un file o dall'altro.

Il programma *BLink* (successore dell'originale ALink), scritto da John Toebes della Software Distillery, è un programma liberamente distribuibile e usato quasi universalmente per la programmazione nei linguaggi C e Assembly. Il comando di link assume, normalmente, una delle seguenti forme. Ancora una volta, non usiamo il percorso completo; sistematelo in modo che si adatti al vostro sistema.

```
(Mcc): Blink OHCE.OBJ lib lib/Amiga.lib to OHCE
(CG):  Blink OHCE.O SmallLib/small.lib to OHCE
(D2):  Blink OHCE.O lib/Amiga.lib to OHCE
(CAPE): Blink67 OHCE.OBJ lib LinkFiles/Amiga.lib to
OHCE
```

I nomi dei file sopracitati sono arbitrari, ma tipici. Ricordate che CAPE produrrà un file oggetto chiamato OHCE.OBJ solo se gli viene chiesto esplicitamente, altrimenti produrrà il file *AsmObjTemp*. Il file *Small.lib* funziona meglio (con la maggior parte di versioni di BLink) se omettiamo la parola chiave LIB (library).

Eseguiamo il risultato

Un comando CLI come OHCE DING DONG produrrà il soddisfacente risultato di GNOD GNID. Abbiamo invertito la coda. Vediamo alcune cose da provare o su cui riflettere. Se redirezioniamo l'output come in

```
OHCE >RAM:XX DING DONG
```

i caratteri di redirezione faranno parte della coda della linea di comando? Se sì, li vedremo apparire nel file di output. Eventuali spazi vuoti supplementari all'inizio o alla fine della linea di comando verranno mantenuti o scartati? Potreste verificare che RUN OHCE DING DONG funziona come ci si aspetterebbe. Potreste anche provare a eseguire uno script file contenente un comando OHCE per verificare che tutto funzioni anche in quella maniera. Infine: se scriviamo OHCE DING DONG, quanti caratteri si troveranno nella coda della linea di comando? Il carattere di newline alla fine della linea verrà incluso nella conta (nel registro D0) oppure no?

Variazioni e documentazione

Potreste avere un assembler diverso da quelli trattati in questo articolo, ma scoprirete che i principi di funzionamento sono gli stessi. Nel caso dobbiate definire i vostri valori di _LVO, le tabelle seguenti vi daranno gli offset in forma decimale, esadecimale ed esadecimale in complemento a due, dei punti di ingresso. Il vostro debugger o assembler potrebbe usare una qualsiasi di queste notazioni.

Non lasciatevi trascinare a comperare tutti i libri di referimento su Amiga che potete trovare... non ancora, perlomeno. Vi costerebbe un bel po' di soldi e occuperebbe parecchio spazio nella vostra libreria. Aspettate di avere deciso in quale area volete investigare nei dettagli. Nel frattempo esiste una comoda forma di documentazione veloce che risulta gratuita per tutti gli utenti Amiga: i file FD.

I file FD si trovano sul vostro disco *Extras*. Sono in formato testo, quindi possiamo stamparli con un comando CLI come TYPE. Essi ci daranno un suggerimento sui campi che bisogna preparare e sui registri che bisogna utilizzare.

$FF22	-$DE	-222	Execute
$FF7C	-$84	-132	IoErr
$FF82	-$7E	-126	CurrentDir
$FFA6	-$5A	-90	UnLock
$FFAC	-$54	-84	Lock
$FFC4	-$3C	-60	Output
$FFCA	-$36	-54	Input
$FFD0	-$30	-48	Write
$FFD6	-$2A	-42	Read
$FFDC	-$24	-36	Close
$FFE2	-$1E	-30	Open

Alcuni punti d'ingresso in *dos.library*

$FDD8	-$228	-552	OpenLibrary
$FE62	-$19E	-414	CloseLibrary
$FE80	-$180	-384	WaitPort
$FE86	-$17A	-378	ReplyMsg
$FE8C	-$174	-372	GetMsg
$FEDA	-$126	-294	FindTask
$FF2E	-$D2	-210	FreeMem
$FF3A	-$C6	-198	AllocMem
$FF76	-$8A	-138	Permit
$FF7C	-$84	-132	Forbid

Alcuni punti d'ingresso in *exec.library*

Gadget multipli

di Peter Booth

Ci sono molte occasioni nelle quali risulta necessario creare una serie di gadget che siano identici fra loro, fatta eccezione per la zona dello schermo nella quale appaiono (e magari anche per del testo eventualmente associato). Potremmo, per esempio, progettare una serie di gadget da usare per selezionare opzioni al posto di usare i menu o per simulare la tastiera di un calcolatore.

In queste occasioni, è una buona idea quella di generare tramite algoritmo la serie di gadget necessari.

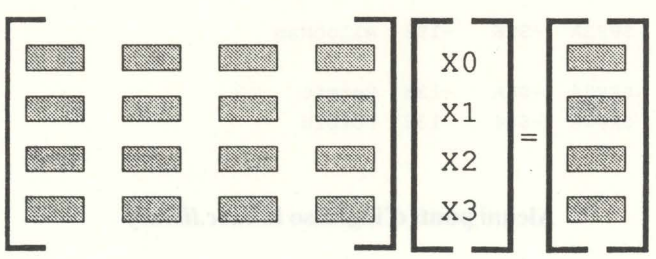
Questo tipo di approccio presenta molti vantaggi. Le dimensioni del codice del programma vengono notevolmente ridotte (principalmente come risultato della perdita di numerose definizioni di strutture di tipo static), le posizioni dei gadget possono essere modificate agevolmente e, naturalmente, noi dobbiamo progettare un solo gadget, senza badare al numero di quelli che vengono richiesti sul display.

È proprio vero che quando bisogna risolvere qualche problema specifico, l'approccio che utilizza un algoritmo al fine di risolverlo risulta spesse volte quello migliore.

Per esempio, potremmo voler creare delle tavole da gioco, o suddividere delle parti del display di fondo trasformandole in sezioni attive, utilizzando gadget senza immagini associate.

Oppure potremmo essere al lavoro su un programma che risolve equazioni simultanee e volere la possibilità di inserire dati numerici direttamente nelle corrispondenti definizioni della matrice. In questo caso potrebbe esserci un gadget per ogni elemento nella matrice e se il programma dovesse essere capace di risolvere diversi numeri di incognite, avremmo bisogno di creare un numero diverso di gadget ogni volta.

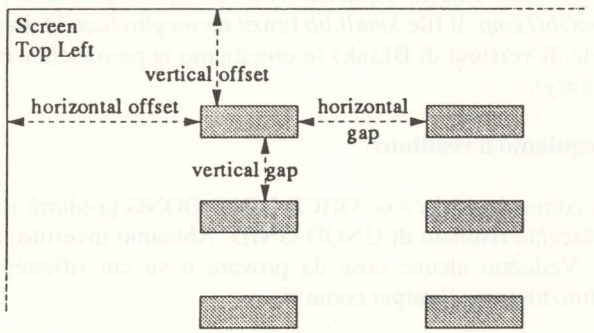
Per esempio, se un utente volesse calcolare quattro incognite, potremmo preparare un display, utilizzando dei gadget, simile a quello mostrato di seguito:



Nonostante molte variazioni sul tema di fondo, il problema è essenzialmente quello di generare in maniera efficiente un 'array di gadget'!

Problemi di questo tipo vengono risolti una volta per tutte trattando il caso generale della produzione di un array di gadget di dimensioni N * M. Per creare una simile collezione di gadget, non dobbiamo solamente considerare l'altezza e la profondità del gadget, ma anche cose come le coordinate di partenza dell'array, gli intervalli orizzontali e verticali fra i gadget e così via.

La figura seguente dovrebbe dare un'idea di una disposizione semplice ma abbastanza soddisfacente:



In pratica

Per creare un singolo gadget dobbiamo allocare della memoria per una struttura Gadget, inizializzare i vari parametri nei campi del gadget e quindi aggiungere il gadget (o meglio la sua struttura) all'apposita lista gestita dalla finestra ospite.

Per preparare N * M gadget di questo tipo, dovremo semplicemente ripetere le operazioni sopra descritte N * M volte. L'unico cambiamento significativo consiste nella scrittura di una routine per preparare e accedere alle strutture Gadget.

Per generare l'array possiamo usare un ciclo a doppia nidificazione basato sul seguente esempio:

```
/************************************************/
for (i=0; condizione i; i++)
    {
```

```
   for (j=0; condizione j; j++)
      {
      Alloca la memoria per il gadget[i,j]
      Inizializza i campi static del gadget
      Inizializza i campi calcolati del gadget
      Aggiungi il gadget[i,j] alla lista
      }
   }
Esegui refresh dei gadget

/*************************************************/
```

Per illustrare l'idea generale, ho scritto del codice che prepara un array di gadget di tipo booleano.

L'esempio non contiene alcun controllo degli errori perché la sua struttura molto semplice ne sarebbe inutilmente complicata.

In pratica, il caso più importante di gestione dell'errore riguarderebbe *AllocMem()* e questo probabilmente verrebbe effettuato al meglio ripercorrendo a ritroso l'array di puntatori, utilizzando una versione a contatore del ciclo a doppio annidamento per liberare tutti i blocchi già allocati.

Un altro tipo di approccio per riservare della memoria potrebbe consistere nell'utilizzo della funzione *AllocRemember()* per ogni allocazione. Si potrebbe anche calcolare la quantità totale di memoria richiesta, allocarla con una sola chiamata ad *AllocMem()* e quindi suddividerla nel numero di strutture Gadget individuali necessarie (questi ultimi due metodi permetterebbero di liberare la memoria già allocata, in qualsiasi momento, mediante una singola chiamata di funzione).

Quali sono i vantaggi?

Tanto per cominciare, una simile organizzazione dei gadget risulta più facile da codificare e poiché questi ultimi vengono posizionati calcolando le loro coordinate, questo approccio si presta a molti possibili usi.

Per esempio, la posizione dei gadget potrebbe essere basata su una semplice somma (come nel nostro esempio) o potrebbe essere ricavata da funzioni più complesse (magari mettendo i gadget attorno a una roulette, calcolando l'appropriato paio di coordinate (x,[f(x)]).

Si potrebbero, anche, creare delle liste di coordinate predeterminate e quindi le posizioni dei gadget verrebbero ottenute leggendo la lista man mano che questi verrebbero creati.

Insomma, questa è la sostanza del nostro metodo. In pratica, è un modo coinciso ed efficace per risolvere questo tipo di problemi.

Ecco il frammento di codice che può aiutarvi a cominciare:

```
/*************************************************/
/*                                               */
/* Costanti e dichiarazioni tipiche              */
/*                                               */
/*************************************************/

#define ROWS    9
#define COLUMNS 9
#define HEIGHT  5
#define WIDTH   20
#define H_GAP   10
#define V_GAP   2
#define H_OFFSET  40
#define V_OFFSET  40

struct Window *window_p;
struct Border border;

/* dichiarazione tipica */

struct Gadget *array[ROWS][CLOUMNS];

/*************************************************/
/*                                               */
/* Esempio di codice                             */
/*                                               */
/*************************************************/

for (i=0; i < ROWS; i++)
   {
   for (j=0; j < COLUMNS; j++)
      {
      array[i][j] = (struct Gadget *)AllocMem(
                           sizeof(struct Gadget), 0);

      array[i][j]->NextGadget = NULL;
    array[i][j]->LeftEdge = WIDTH*j+H_GAP*j+H_OFFSET;
    array[i][j]->TopEdge = HEIGHT*i+V_GAP*i+V_OFFSET;
      array[i][j]->Width = WIDTH;
      array[i][j]->Height = HEIGHT;
      array[i][j]->Flags = GADGHCOMP;
      array[i][j]->Activation = RELVERIFY;
      array[i][j]->GadgetType = BOOLGADGET;
      array[i][j]->GadgetRender = (APTR)&border;
      array[i][j]->SelectRender = NULL;
      array[i][j]->GadgetText = NULL;
      array[i][j]->MutualExclude = NULL;
      array[i][j]->SpecialInfo = NULL;
      array[i][j]->GadgetID = (i<<8)+j;
      array[i][j]->UserData = NULL;

      AddGadget(window_p, array[i][j], 0);
      }
   }
RefreshGadgets(window_p->FirstGadget, window_p,
NULL);

/*************************************************/
```

L'arte della programmazione in linguaggio Assembly

18 modi di rendere il vostro codice macchina più corto e più veloce

di John Toebes

John Toebes è analista di sistemi presso il SAS Institute Inc. a Cary, North Carolina. Ha programmato su Amiga fino dal 1985 ed è l'autore delle recenti versioni del compilatore Lattice C. Coordina un gruppo chiamato The Software Distillery, responsabile di molti programmi liberamente distribuibili che hanno acquisito larga popolarità come Hack, PopCLI, MemWatch e Blink. John può essere raggiunto su BIX (JTOEBES) e sulla BBS della Software Distillery al numero USA 919-471-6436.

Si dice spesso che se vuoi fare qualcosa di veloce, devi farlo in linguaggio macchina. Sebbene ci sia della verità in questa affermazione, è più importante che il programma sia scritto bene, non importa in quale linguaggio. Con il linguaggio assembly, in modo particolare, è possibile scrivere del codice pessimo o inefficiente senza nemmeno accorgersene. Solamente imparando a riconoscere le cattive abitudini di programmazione, si può sperare di imparare a scrivere del codice migliore.

In un lasso di tempo molto ridotto, un programmatore esperto di linguaggio assembly può esaminare un frammento di codice e classificarlo come:

```
1) Eccezionale
2) Buono
3) Qualsiasi cosa non valga la pena di classificare
```

Non esistono regole precise e veloci che permettano di classificare questi tipi di programmi, ma esistono certe linee direttrici alle quali ci si può riferire. Nonostante queste linee di principio tendano a differire ampiamente da persona a persona, quando il codice in esame è 'eccezionale' o 'buono', raramente ci si trova in disaccordo su queste classificazioni. L'intento di questo articolo è quello di aiutarvi a scrivere del codice che venga riconosciuto per i suoi meriti.

Eccezionale - tende a mettersi in mostra abbastanza facilmente, utilizzando quasi tutti i trucchi conosciuti (inventandone spesso di nuovi lungo il percorso) ed è organizzato in modo che il codice vada dritto al punto.

Buono - tende a essere abbastanza facile da seguire, senza errori vistosi o mancate ottimizzazioni, ma spesso perde terreno dal punto di vista dell'organizzazione.

Dall'altra parte, il codice *cattivo* tende a essere, beh... non così buono.

Possiamo dividere la strategia per scrivere del codice ottimale in tre aree base:

```
1  ottimizzazioni a livello di istruzioni (peephole)
2  ottimizzazioni locali
3  ottimizzazioni globali
```

In realtà, se questi termini sembrano derivare dalla tecnologia dei compilatori, è perché lo sono effettivamente. I compilatori devono passare attraverso simili fasi per decidere riguardo al codice da generare, sebbene essi non sempre dispongono della quantità di informazioni disponibile invece al programmatore assembly.

Ottimizzazioni a livello di istruzioni

Nel campo della scelta delle istruzioni vere e proprie, la famiglia 68000 offre una vasta serie di scelte alternative con il suo numeroso insieme di comandi e di modi di indirizzamento. Concentreremo la nostra attenzione sul 68000 in quanto è il processore più popolare e perché le ottimizzazioni che vedremo qui si possono applicare anche ai processori più recenti. Alcune delle linee di riferimento che possiamo seguire nello scegliere le istruzioni da utilizzare sono:

1 Evitare il modo di indirizzamento immediato a 32 bit

2 Evitare operandi a 32 bit con i registri indirizzi

3 Evitare i modi di indirizzamento assoluto a 32 bit

4 Evitare le istruzioni MUL/DIV

5 Usare la forma QUICK delle istruzioni, quando possibile

6 Dimenticare che l'istruzione CLR esiste

Vediamo come possiamo applicare queste regole con qualche semplice istruzione:

```
L1:   move.l  #1,D0          6 byte  12 cicli
L2:   add.l   #2,A0          6 byte  16 cicli
L3:   move.l  #3,Datavar    10 byte  28 cicli
L4:   muls.w  #10,D0         4 byte  70 cicli
L5:   clr.l   D0             2 byte   6 cicli
```

Queste operazioni possono essere effettuate più efficacemente dalle seguenti istruzioni:

```
L1:   moveq   #1,D0          2 byte   4 cicli
L2:   addq.w  #2,A0          2 byte   8 cicli
L3a:  moveq   #3,D0          2 byte   4 cicli
      move.l  D0,Datavar     6 byte  20 cicli
                [totale:     8 byte  24 cicli]
L3b:  moveq   #3,D0          2 byte   4 cicli
      move.l  D0,Datavar(A4) 4 byte  16 cicli
                [totale:     6 byte  20 cicli]
L4:   move.l  D0,D1          2 byte   4 cicli
      lsl.l   #2,D1          2 byte  12 cicli
      add.l   D1,D0          2 byte   8 cicli
      add.l   D0,D0          2 byte   8 cicli
                [totale:     8 byte  32 cicli]
L5:   moveq   #0,D0          2 byte   4 cicli
```

Come potete vedere da queste semplici istruzioni, non sempre si può fare una modifica che consenta una miglioria su tutti i fronti. Nell'esempio L4 abbiamo raddoppiato le dimensioni del codice ma in cambio abbiamo ottenuto di dimezzarne i tempi di esecuzione. Dal momento che l'aumento delle dimensioni è così piccolo, questa è una situazione particolare nella quale è ragionevole generare più codice, dal momento che il risparmio di tempo che ne deriva vale di più.

Con L3 abbiamo un eccellente esempio di come, generando due istruzioni anziché una, si ottiene un risparmio sia di tempo che di dimensioni del codice.

Questo fatto richiama una regola importante:

7 Se potete usare MOVEQ per una costante a 32 bit, *fatelo*!

Questa regola si applica a qualsiasi istruzione con la quale si può usare un registro dati. Per esempio:

```
and.l   #15,D2          6 byte  16 cicli
or.l    #2,D3           6 byte  16 cicli
sub.l   #28,D4          6 byte  16 cicli
cmp.l   #1,D1           6 byte  14 cicli
                [totale: 24 byte  62 cicli]
```

diventerebbe parecchio più veloce e corto se scritto così:

```
moveq   #15,D0          2 byte   4 cicli
and.l   D0,D2           2 byte   8 cicli
moveq   #2,D0           2 byte   4 cicli
or.l    D0,D3           2 byte   8 cicli
moveq   #28,D0          2 byte   4 cicli
sub.l   D0,D4           2 byte   8 cicli
moveq   #1,D0           2 byte   4 cicli
cmp.l   D0,D1           2 byte   8 cicli
                [totale: 16 byte  48 cicli]
```

Ma aspettate, non dovete per forza usare sempre un registro. Qualche volta si vuole sottrarre un piccolo numero da un registro, ma il numero è troppo grande per essere usato con l'istruzione SUBQ.

In questa situazione, è abbastanza ragionevole usare due istruzioni SUBQ di seguito, in modo che:

```
sub.l   #10,D0          6 byte  16 cicli
```

diventi:

```
subq.l  #8,D0           2 byte   8 cicli
subq.l  #2,D0           2 byte   8 cicli
                [totale: 4 byte  16 cicli]
```

Un'altra situazione comune dove le costanti immediate giocano un ruolo importante è la manipolazione dello stack. Quando si chiama una routine C che richiede il passaggio di parametri sullo stack, l'uscita da quella routine di solito appare così:

```
add.l   #12,SP          6 byte  16 cicli
```

Se il numero di byte da aggiungere è uguale o inferiore a otto, dovremmo utilizzare invece:

```
addq.w  #8,SP           2 byte   8 cicli
```

Per nove o più byte possiamo utilizzare l'istruzione LEA:

```
lea     12(SP),SP       4 byte   8 cicli
```

Infine, non dobbiamo limitarci all'istruzione MOVEQ per caricare costanti nei registri. A seconda delle costanti, ci sono un numero di trucchi che si possono usare per caricare un valore. Per esempio, per caricare il valore $10000 in un registro, potreste essere stati tentati di scrivere:

```
move.l  #$10000,D0      6 byte  12 cicli
```

Un programmatore acuto, invece, scriverebbe:

```
moveq   #1,D0           2 byte   4 cicli
swap    D0              2 byte   4 cicli
                [totale: 4 byte   8 cicli]
```

Chiaramente questo funziona per qualsiasi numero nell'intervallo tra $FF800000 e $007F0000, la cui word inferiore è tutta a zero. Un altro trucco di questo genere consiste nel caricare una costante che è troppo grossa per essere usata con una istruzione MOVEQ:

```
moveq   #200-256,D0     2 byte   4 cicli
neg.b   D0              2 byte   4 cicli
            [totale:    4 byte   8 cicli]
```

In effetti, qualsiasi valore a 8 bit può essere caricato in un registro dati in questa maniera, utilizzando MOVEQ/NEG.B. Un trucco ancora più furbo, che si incontra però meno frequentemente, consiste nel fare seguire l'istruzione MOVEQ da NEG.W per generare costanti del tipo $FFFF00xx e $0000FFxx.

Con le istruzioni ROL/ROR si possono caricare singoli bit al costo di un po' di velocità. Per esempio, per generare un numero compreso fra $80000000 e $01000000 possiamo scrivere:

```
moveq   #1,D0           2 byte   4 cicli
ror.l   #n,D0           2 byte   8+2n cicli
```

oppure, per generare un numero compreso fra $00000080 e $00004000 possiamo scrivere:

```
moveq   #$40,D0         2 byte   4 cicli
rol.l   #n,D0           2 byte   8+2n cicli
```

Con questi trucchi, si possono generare praticamente tutte le costanti che si incontrano più frequentemente, utilizzando solo quattro byte anziché i normali sei. Tutto questo, a lungo andare, si somma e fa peso.

Le ottimizzazioni viste finora erano rivolte a registri dati, ma ci sono anche alcuni trucchi per i registri indirizzi:

8 Usate la forma .W delle istruzioni per i registri indirizzi, quando possibile

9 Se potete, eseguite le operazioni nei registri indirizzi

Per esempio, se avete bisogno di caricare un valore costante in un registro indirizzi, potreste avere scritto:

```
move.l  #502,A1         6 byte  12 cicli
```

Possiamo, invece, ingannare l'istruzione LEA per farle fare lo stesso lavoro:

```
lea     502.W,A1        4 byte   8 cicli
```

Naturalmente, se vogliamo caricare uno zero in un registro indirizzi possiamo fare anche meglio scrivendo:

```
sub.l   A1,A1           2 byte   8 cicli
```

La forma .W di un'istruzione è ancora più utile per spingere costanti sullo stack in preparazione alla chiamata di una routine C. Il codice che ci aspetteremmo in questo caso sarebbe:

```
move.l  #1,-(SP)        6 byte  20 cicli
```

oppure, adesso che sappiamo fare un po' meglio:

```
moveq   #1,D0           2 byte   4 cicli
move.l  D0,-(SP)        2 byte  12 cicli
            [totale:    4 byte  16 cicli]
```

Possiamo, invece, usare l'istruzione PEA:

```
pea     1.W             4 byte  16 cicli
```

Questo non sembra un grande risparmio per quelle piccole costanti che MOVEQ può gestire, comunque lo diventa per quelle costanti che stanno in 16 bit ma che sono troppo grosse per MOVEQ. Sul 68020 e 68030 l'esecuzione dell'istruzione MOVEQ è ancora più veloce per le costanti, grazie a delle pipeline speciali contenute nel processore, quindi conviene sempre utilizzarla quando possiamo.

Su Amiga, un'operazione su registri indirizzi che viene effettuata abbastanza spesso è la conversione di BPTR in APTR. Dal momento che un BPTR è spostato sulla destra di due bit, uno sarebbe tentato di usare un registro dati per eseguire l'operazione (perché le istruzioni di shift e rotate non operano sui registri indirizzi).

Anche il trasferimento dell'operazione in un registro dati risulta costoso:

```
exg     A0,D0           2 byte   6 cicli
lsl.l   #2,D0           2 byte  12 cicli
exg     D0,A0           2 byte   6 cicli
          [totale:      6 byte  24 cicli]
```

Usando l'istruzione MOVE anziché la EXG riusciamo a risparmiare solo quattro cicli. Possiamo fare molto meglio, invece, usando due istruzioni:

```
  add.l   A0,A0         2 byte   8 cicli
  add.l   A0,A0         2 byte   8 cicli
            [totale:    4 byte  16 cicli]
```

Sicuramente ci sono altre ottimizzazioni a livello di istruzioni che si possono applicare con il 68000. In ogni caso, queste sono le più comuni e utili per del codice normale.

Un metodo eccellente per impararle consiste nel procurarsi un buon manuale Motorola e leggere le temporizzazioni delle istruzioni. Alcune di esse sono assai sorprendenti.

Queste ottimizzazioni, comunque, sono in sé solo una piccola parte di quello che ci rende buoni programmatori assembly (o *eccezionali*, come vorremmo essere). In realtà si possono ottenere miglioramenti enormemente più consistenti cambiando gli algoritmi utilizzati dal nostro codice, piuttosto che effettuando semplici sostituzioni di istruzioni.

Per questa ragione, esamineremo ora una serie di possibili ottimizzazioni locali.

Ottimizzazioni locali

A differenza del tipo precedente, l'ottimizzazione locale richiede di pensare più a lungo all'organizzazione delle cose e spesso produce pochi risultati sull'algoritmo che stiamo tentando di implementare. Il più semplice che viene in mente è quello usato nelle operazioni che corrispondono a una moltiplicazione di un linguaggio di alto livello.

Per esempio, per accedere con un indice (index) in una serie (array) di oggetti, è necessario moltiplicare l'indice per le dimensioni dell'operando. In assembly, un buon programmatore lavora con dimensioni che siano potenze di due.

Invece di:

```
move.l  index,D0
muls.w  #4,D0
```

possiamo scrivere:

```
move.l  index,D0
lsl.l   #3,D0
```

Sull'argomento degli spostamenti (shift), ci sono delle semplici regole da seguire:

10a Se il numero da spostare è più grande di 15, usate l'istruzione SWAP e quindi sottraete 16 dal numero da spostare

10b Se il numero è uguale o maggiore di 8, generate uno spostamento di otto e quindi sottraete 8 dal numero da spostare

10c Se il numero è 1 e lo spostamento è verso sinistra, generate un'istruzione ADD al suo posto

10d Se il numero è maggiore di 0, generate uno spostamento di quelle dimensioni

Prendendo queste regole e applicandole, otteniamo le seguenti sequenze:

```
add.l   D0,D0   ; shift a sinistra di 1
lsl.l   #2,D0   ; shift a sinistra di 2
lsl.l   #4,D0   ; shift a sinistra di 4
lsl.l   #8,D0   ; shift a sinistra di 8

lsl.l   #8,D0   ; shift a sinistra di 9
add.l   D0,D0

lsl.l   #8,D0   ; shift a sinistra di 10
lsl.l   #2,D0

swap    D0      ; shift a sinistra di 16
clr.w   D0

swap    D0      ; shift a sinistra di 24
clr.w   D0
lsl.l   #8,D0

swap    D0      ; shift a sinistra di 25
clr.w   D0
lsl.l   #8,D0
add.l   D0,D0

lsr.l   #1,D0   ; shift a destra di 1 (non
ottimizzato)

clr.w   D0      ; shift a destra di 16
swap    D0

clr.w   D0      ; shift a destra di 25
swap    D0
lsr.l   #8,D0
lsr.l   #1,D0
```

Tutte queste sequenze risultano essere migliori del fatto di caricare una costante di shift in un registro e del fare lo shift, in quanto usano un registro in meno e sono più veloci. Questo vale in modo particolare per l'intervallo 9-15. Quando generate uno shift di 16, ricordate sempre di mettere SWAP al posto giusto.

È importante notare che questi shift non sono molto attenti al bit di segno, ma in assembly questo è raramente importante. Se dobbiamo gestire uno shift con segno di 16 bit o più, possiamo usare l'istruzione EXT.L anziché la CLR.W.

Una ottimizzazione particolarmente bizzarra che risulta molto utile quando si scrive del codice assembly *in assembly*, sfrutta il fatto che i bit vengono scartati mentre lo spostamento sta avendo luogo. Supponiamo, per esempio, che nel generare il campo destinazione di un registro appartenente a un'istruzione, sia necessario mascherare il valore con un 7 e quindi spostarlo a sinistra di 9. Un buon programmatore assembly avrebbe scritto (assumendo che il valore fosse già in D0):

```
moveq   #7,D1          2 byte   4 cicli
and.l   D1,D0          2 byte   8 cicli
lsl.l   #8,D0          2 byte  24 cicli
add.l   D0,D0          2 byte   8 cicli
                [totale:  8 byte  44 cicli]
```

Ma un programmatore più astuto avrebbe notato la possibilità che si presenta qui (sapendo che il valore del registro può essere contenuto in un byte) e avrebbe scritto:

```
lsl.l   #5,D0          2 byte  16 cicli
lsl.l   #4,D0          2 byte  16 cicli
            [totale:   4 byte  32 cicli]
```

11 Tenete estremamente d'occhio i codici di condizione (CC)

Un *eccezionale* programma in assembly non usa quasi mai l'istruzione TST. Il 68000 è molto bravo nel modificare il registro dei codici di condizione per riflettere praticamente qualsiasi operazione effettuata. Solo quando mettiamo qualco-

sa in un registro indirizzi questi codici vengono lasciati inalterati.

Questa caratteristica, comunque, può ritorcersi contro di noi se aggiungiamo semplicemente del codice a un programma già esistente, senza controllare i condition codes, in quanto praticamente qualsiasi cosa può distruggerli. Può essere utile ricordare che l'istruzione MOVEM, sebbene sia più voluminosa e più lenta dell'istruzione MOVE, può tranquillamente leggere o scrivere in un singolo registro senza alterare i condition codes.

È una buona regola empirica quella di dimenticare l'esistenza dell'istruzione TST (se possibile) e di sistemare il codice in modo che i condition codes vengano settati senza intervento esplicito.

Tenere d'occhio i condition codes risulta molto utile quando si deve prendere più di una decisione. I programmatori assembly si possono permettere il lusso di verificare tre condizioni alla volta:

```
add     val1,D0  ; fai qualche operazione per
                 ; settare il registro CC (non una TST)
blt.s   negativo ; se negativo, dobbiamo fare
                 ; qualche altra cosa
beq.s   done     ; se è zero abbiamo finito
...              ; oppure arriviamo qui con un
                 ; valore positivo
```

Nella maggior parte dei linguaggi di alto livello (con l'eccezione del Fortran) questo tipo di situazione non si incontra quasi mai, ma nei buoni programmi assembly, i valori e i flag sono messi in modo da sfruttare la situazione.

Quando scegliamo le istruzioni di branch, è importante essere consapevoli di quanto spesso i branch saranno verosimilmente utilizzati. Le condizioni di branch dovrebbero essere organizzate in modo che *non* vengano effettuati salti la maggior parte delle volte.

Il rilevamento di situazioni di errore dovrebbe essere effettuato da un test che salta eventualmente fuori dal flusso principale, permettendo nei casi normali di continuare l'esecuzione dall'istruzione seguente quella di branch. In questo modo, nei processori di tipo più recente non si interrompe la pipeline e anche sul 68000 si trae il vantaggio di un tempo di esecuzione ridotto dal fatto che il salto non viene effettuato:

```
moveq   #1,D0
and.l   D0,D1  ; è settato il flag di errore?
bne.s   error1 ; si, segnalare errore
...            ; continuare qui nella maggior parte
               ; dei casi
```

12 *Abusare* dell'istruzione DBcc

La migliore istruzione di tutte, comunque, è la serie DBcc. Anche solo sul 68010 questa istruzione mette il processore in uno stato speciale durante il quale l'istruzione contenute nel loop viene tenuta in memoria cache per un'esecuzione rapida e ripetuta.

L'istruzione DBRA, ad ogni modo, presenta anche alcuni importanti svantaggi di cui bisogna essere consapevoli:

12a Il contatore di loop è solo di 16 bit. La parte superiore del registro rimane inalterata.

12b Per migliorare le prestazioni, dovreste includere una sola istruzione nel loop.

12c I loop nei quali si entra dall'inizio sono basati su *zero*, quindi vengono eseguiti una volta di più rispetto al numero contenuto nel registro.

Anche con queste limitazioni, l'istruzione DBRA può essere usata in molte situazioni. La più frequente è nella copia di zone di memoria:

```
     moveq   #15,D0 ; copia 16 byte di memoria
                    ; partendo da qualsiasi
; indirizzo (pari/dispari)

     move.l  source,A0
     move.l  dest,A1
lab: move.b  (A0)+,(A1)+
     dbra    D0,lab
```

Se non conosciamo la lunghezza prima del tempo, si può rimediare entrando nel loop in corrispondenza dell'istruzione DBRA:

```
     move.w  len,D0
     move.l  source,A0
     move.l  dest,A1
     bra.s   dbr
lab: move.b  (A0)+,(A1)+
dbr: dbra    D0,lab
```

In ogni caso, le istruzioni della famiglia DBcc sono troppo potenti per lasciarle solamente a copiare byte. Sono eccezionali per comparare e riempire la memoria e anche per cercare caratteri. Il trucco sta solo nell'usare i condition codes appropriati per uscire dal loop.

Per esempio, se volessimo cercare un carattere in una stringa, potremmo scrivere:

```
     move.w  len,D0
     move.l  string,A0
     moveq   #char,D1  ; carattere da cercare
     bra.s   dbr
lab: cmp.b   (A0)+,D1
dbr: dbne.s  D0,lab
```

In questo caso abbiamo dovuto fare attenzione. La lunghezza potrebbe anche essere stata zero, quindi dovevamo iniziare in fondo al loop. Il loop, comunque, termina quando i condition codes risultano uguali a EQ, quindi dobbiamo essere sicuri che l'ultima cosa che facciamo prima di iniziare il loop lasci un va-

lore diverso da EQ nei CC. Da qui l'uso di MOVEQ prima di BRA.

Ottimizzazioni globali

Le ottimizzazioni locali e a livello di istruzione che abbiamo visto ci permettono di sistemare i punti caldi del codice e in generale di rendere più veloci i casi microscopici. Per avere il maggiore ritorno dei propri sforzi, comunque, esistono svariate ottimizzazioni globali vincenti:

13 Concedersi il tempo per scegliere accuratamente i registri

Un buon programma tenta di assegnare i registri agli oggetti di grande utilizzo o anche a quelli che risultano più importanti in un particolare istante. Il segreto per scrivere un programma eccezionale sta nel sistemare il codice e le scelte dei registri in modo che il valore che serve sia sempre nel registro giusto. In particolare, è desiderabile evitare di dover usare l'istruzione MOVE da registro a registro solo per mettere qualcosa al posto giusto per una chiamata.

14 Evitare l'istruzione MOVEM, se possibile

Siccome lo standard delle chiamate su 68000 richiede di preservare il contenuto di tutti i registri eccettuati D0/D1/A0/A1, è necessario salvare ogni altro registro utilizzato nel punto di entrata e ripristinarlo prima dell'uscita. È assai desiderabile che si scelgano i registri in modo che se ne debbano salvare il numero minore possibile. Se dobbiamo salvare il contenuto di un solo registro, allora è consigliabile usare l'istruzione MOVE al posto della MOVEM.

15 Evitare le istruzioni LINK e UNLK

Mentre LINK è spesso usata nei linguaggi di alto livello, il suo uso in programmi assembly risulta abbastanza raro e principalmente utilizzato per creare delle zone di memorizzazione temporanea per elementi troppo grandi per entrare in un registro. Se non ci sono abbastanza registri da usare per un'operazione, formandosi quindi l'esigenza di usare parte dello stack per la memorizzazione, potrebbe risultare più conveniente creare una subroutine per una parte dell'operazione (se il tempo non è un fattore critico) o potrebbe addirittura risultare più conveniente spingere i parametri in eccedenza sullo stack, anziché memorizzarli in una locazione fissa della zona allocata con LINK.

16 Fare scelte intelligenti per flag e valori

Se abbiamo tre possibili stati per un flag, allora scegliamo -1, 0 e 1 in modo da poter utilizzare un branch a tre vie. Se il valore verrà utilizzato come indice di una tabella, scegliamo quelli *pari* che corrispondano direttamente all'indice degli elementi in modo da evitare una moltiplicazione. Per quanto riguarda i *bit*, mettiamo quello più usato nella posizione del segno, cosicché il singolo bit viene settato semplicemente come effetto collaterale dell'azione di caricare i flag. Se utilizziamo una *long word*, abbiamo a disposizione 3 bit che possono influenzare il bit di segno a seconda della lunghezza (byte, word, long word) sulla quale viene effettuata l'operazione; usiamoli per primi.

17 Usate le tabelle, quando è il caso

Molte operazioni possono essere svolte rapidamente facendo uso di tabelle. Se, per esempio, stiamo effettuando delle trasformazioni per una rotazione su schermo, potrebbe essere sufficiente pre-calcolare il seno per tutti i 360 gradi, arrotondare l'angolo di rotazione desiderato al più vicino grado intero e quindi consultare direttamente la tabella, anziché fare calcoli elaborati. Le tabelle di salto per svolgere determinate operazioni (specialmente quando il numero per l'operazione è scelto nel modo intelligente come visto sopra) risultano particolarmente veloci.

L'artiglieria pesante

Il numero di opportunità che si presentano per effettuare delle ottimizzazioni sul 68000 è infinito. Questa non è assolutamente una lista completa. In ogni caso, il tempo a disposizione per operare le ottimizzazioni è finito. Per questo motivo, è importante misurare l'importanza di una particolare ottimizzazione rispetto al tempo necessario per implementarla.

Un eccellente strumento d'aiuto di cui bisogna prendere in considerazione l'uso è il profiler. Questo ci dirà in quale parte del codice viene speso il tempo maggiore e ci permetterà di concentrarci su questa zona.

Una buona regola empirica stabilisce che il 95% del tempo viene speso in circa il 5% del codice. Si trae poco beneficio nell'ottimizzare il rimanente 95%, se non si è prestata particolare attenzione a quel *rovente* 5%.

Un ultimo trucco

18 Imparare a usare il bit X

Il bit X è uno dei più strani fra quelli presenti nell'architettura del 68000. Viene modificato da poche istruzioni, ma possiede la proprietà di permetterci di mettere un'intera long word a 0 o -1 basandoci sul suo valore:

```
subx.l  D0,D0
```

Con questo trucco in mente, verificate la vostra abilità nell'ottimizzare il codice macchina per 68000, trovando la migliore sequenza di istruzioni per svolgere il seguente compito: dato un valore in D0 e i condition codes in uno stato sconosciuto, settare D1 a 0 se D0 contiene uno 0 oppure a 1 altrimenti. Potete distruggere il contenuto di D0, se necessario. La risposta si trova esattamente qui sotto, quindi evitate di leggere oltre se volete raccogliere la sfida!!!

Soluzione al problema posto da John Toebes:

```
neg.l   D0
subx.l  D1,D1
neg.l   D1
```

SoftMail

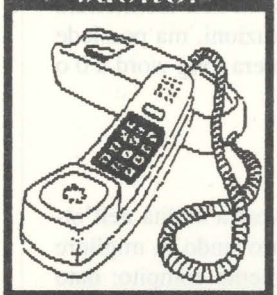
Vendita per corrispondenza di programmi originali ed accessori per computers.

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

® SoftMail è un marchio registrato da Lago snc

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

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



031/30.01.74

ACCESSORI

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

LIBRI/HINTS & TIPS

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

AMIGA

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

COMMODORE 64 CASS.

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

COMMODORE 64 DISCO

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

COMMODORE 128 DISCO (128K, 80 COL.)

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

OFFERTE SPECIALI

(FINO AD ESAURIMENTO)

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

DEFENDER OF THE CROWN C64: cass. 12.000 disco 15.000

... SOFTMAIL TI DA' DI PIU'.

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

Desidero ricevere i seguenti articoli:

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

☐ Pagherò al postino in contrassegno

Titolo del programma	Computer	Cassetta/disco/accessorio	Prezzo

Contributo spese di spedizione Lit. 5.000

TOTALE LIT. _____

ORDINE MINIMO LIT. 20.000 (SPESE ESCLUSE)

Cognome e nome _____

Indirizzo _____ Nr. _____

CAP _____ Città _____ Prov. _____ Telefono _____

FIRMA (SE MINORENNE QUELLA DI UN GENITORE)

VERRANNO EVASI SOLO GLI ORDINI FIRMATI _____

TA006

Disegnare con la libreria grafica

di Romano Tenca

La libreria grafica dell'Amiga ("graphics.library"), contenuta nella ROM della macchina, è una delle più ampie dell'intero sistema: essa comprende infatti più di un centinaio di routine, fra funzioni e macro, che non solo interessano le operazioni del 68000, ma permettono il controllo anche del copper e del blitter, i due coprocessori dedicati fra le altre cose alla gestione della grafica.

Le routine, il cui elenco quasi completo si trova alla fine dell'articolo, possono essere divise in sei gruppi:

- visualizzazione
- uso diretto del copper
- disegno
- uso diretto del blitter
- animazione
- gestione caratteri

Inoltre, la "layers.library" ha dei forti legami con la libreria grafica, tanto che l'"Amiga ROM Kernel Reference Manual: Libraries and Devices" le tratta congiuntamente.

Enorme appare, dunque, la mole della libreria e l'insieme delle conoscenze necessarie a dominarla nella sua interezza (delle 500 pagine di testo del volume citato, più di 200 sono dedicate a questi argomenti, e notevoli sono le lacune e gli errori); pertanto focalizzeremo la nostra attenzione sulle funzioni primitive di disegno e sulla gestione dei colori. Della visualizzazione della pagina grafica lasceremo, nei programmi d'esempio, che si occupi Intuition: con la sua gestione di schermi e finestre, essa costituisce infatti una interfaccia con la libreria grafica che solleva l'utente dal gravoso compito di pilotare direttamente le funzioni di visualizzazione. Tuttavia non è possibile spiegare convenientemente l'uso delle routine di disegno senza accennare al modo in cui l'Amiga opera per visualizzare i dati grafici, e da ciò partiremo dopo aver spiegato come aprire la libreria grafica.

Apriamo e chiudiamo la libreria

Come ogni libreria, anche la libreria grafica va aperta per poter localizzare l'indirizzo di partenza della struttura Library prima della quale si trova la tavola dei salti che permette di accedere alle varie funzioni:

```
GfxBase = (struct GfxBase *)
        OpenLibrary( "graphics.library",0L );
```

Se il valore di ritorno è nullo, allora si è verificato un errore e la libreria non risulta accessibile. Questo non dovrebbe mai accadere, essendo la libreria grafica residente in ROM e non su disco come accade per altre librerie. Tuttavia non si può essere mai sicuri di niente in un sistema multitasking come Amiga; qualche altro programma, per esempio, avrebbe potuto modificare la funzione Exec OpenLibrary (detto per inciso, è quello che fa un virus della nuova generazione con la funzione Exec OldOpenLibrary, la funzione di apertura delle librerie ormai desueta). Inoltre, in futuro, la libreria grafica potrebbe essere implementata su disco e in tal caso il dischetto corrispondente dovrebbe essere nel drive affinché la funzione possa avere esito positivo.

Come viene aperta, così deve essere chiusa dopo l'uso:

```
CloseLibrary( GfxBase );
```

anche qui valgono le medesime considerazioni fatte per l'apertura.

Due parole su copper e ViewPort

Il copper è un coprocessore estremamente veloce che opera in sincronia con il pennello elettronico che disegna il video. In generale si può dire che il copper accetti questo tipo di istruzioni: attendi la tale posizione del pennello elettronico e modifica il valore contenuto nel tale registro hardware. Tutti i registri hardware sono pilotabili attraverso il copper, ma per quanto riguarda la libreria grafica, ovviamente, i principali registri interessati sono quelli che riguardano i colori, la risoluzione, il blitter e la posizione della pagina grafica. La libreria grafica mette a disposizione una serie di funzioni che permettono di guidare il copper in modo del tutto trasparente, semplicemente definendo delle strutture per poi passarle al sistema. Le strutture interessate sono chiamate: View, ViewPort, ColorMap, RasInfo e BitMap. È attraverso di esse che si possono definire le risoluzioni del video, la mappa dei colori e la posizione della pagina grafica in memoria; la libreria grafica si incarica per noi di trasformare i valori ivi contenuti in un programma eseguibile dal copper (da questo punto di vista si comporta quasi come un compilatore).

La View contiene una lista di ViewPort, ognuna delle quali corrisponde a quello che sotto Intuition si chiama schermo; quando, infatti, si apre uno schermo, non si fa altro che definire, indirettamente, una nuova ViewPort da visualizzare attraverso il copper. Così se noi abbiamo aperto una finestra su uno schermo sotto Intuition, aprendone la libreria, possiamo ottenere l'indirizzo della associata ViewPort chiamando la seguente funzione della "intuition.library":

```
vp = ViewPortAddress( w );
```

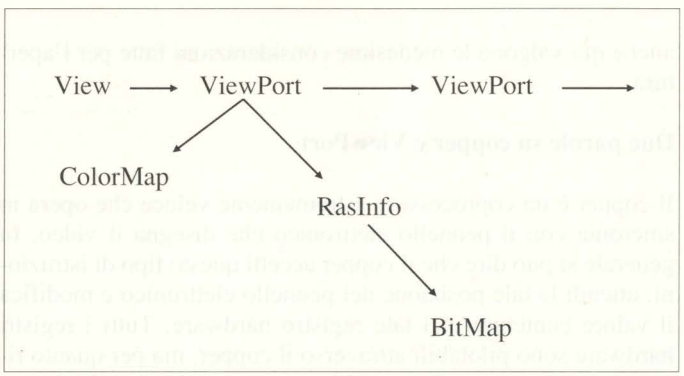
ove "w" e un puntatore alla struttura Window; oppure possiamo ricavare lo stesso valore dall'indirizzo "s" della struttura Screen:

```
vp = &s->ViewPort;
```

si noti come la struttura ViewPort sia fisicamente contenuta nella struttura Screen e come, quindi, sia necessario ricavare l'indirizzo della variabile "s->ViewPort" per ottenere un puntatore alla ViewPort; oppure ancora dall'indirizzo della Window è possibile risalire a quello dello Screen che la contiene e da questo ricavare il puntatore alla ViewPort:

```
vp = &w->WScreen->ViewPort;
```

Ogni ViewPort contiene un puntatore alla struttura RasInfo, la quale, a sua volta, contiene un puntatore alla struttura BitMap, in cui si trovano le informazioni relative alla posizione della pagina grafica in memoria. La ViewPort contiene anche un puntatore alla struttura ColorMap che definisce i colori dello schermo; e proprio dai colori conviene partire per capire il funzionamento della gestione grafica di Amiga.



I colori

L'Amiga può visualizzare 4096 colori: ognuno di essi viene ottenuto mediante la combinazione di determinate intensità dei colori fondamentali: il rosso ("Red"), il verde ("Green"), il blu ("Blue"); di qui deriva appunto il termine "RGB". Ogni colore fondamentale può assumere 16 diverse intensità (tutti i valori compresi tra 0 e 15, esprimibili, dunque, con 4 bit) e le loro combinazioni assommano a 16 * 16 * 16 cioè 4096.

Quando, per esempio, creiamo una nuova ViewPort a 4 colori dobbiamo scegliere quali, tra i possibili 4096, potranno essere usati per disegnare. Ciò avverrà stabilendo il livello di rosso, verde, e blu e ponendo i valori corrispondenti rispettivamente nei bit 8-11, 4-7, 0-3 di una UWORD (unsigned short) che troverà posto nell'array indirizzato da "ColorMap.ColorTable". Nella tavola che segue appare un esempio che prevede quattro colori, i valori sono esadecimali:

```
livello dei colori fondamentali          |array di UWORD
colore |  rosso    verde     blu     | n.|  valore
-------|---------|---------|---------|---|---------
nero   |    0    |    0    |    0    | 0 |   0000
verde  |    0    |    F    |    0    | 1 |   00F0
blu    |    0    |    0    |    F    | 2 |   000F
bianco |    F    |    F    |    F    | 3 |   0FFF
```

Va notato che tale formato potrà cambiare in seguito a mutamenti del sistema operativo e che quindi, per accedere a dati di questo tipo, conviene sempre seguire la strada indicata dai costruttori, utilizzando le previste funzioni della libreria grafica. Per caricare una nuova tavola dei colori è disponibile la funzione:

```
LoadRGB4( vp, rgb, n )
```

in cui "vp" è un puntatore alla ViewPort, "rgb" è un puntatore a un array di UWORD il cui senso è quello appena sopra specificato e "n" indica il numero di elementi che contiene. Tale funzione caricherà nella ColorMap "n" valori RGB a partire dal colore 0. Il numero "4" con cui termina il nome della funzione si riferisce al fatto che 4 sono i bit significativi dei livelli di rosso, verde e blu e che quindi quei valori, come abbiamo già visto, devono essere compresi tra 0 e 15 (0-F hex). Quando si modificano dei colori con questa funzione, tutti i pixel della ViewPort che li visualizzano assumeranno la nuova combinazione stabilita. Per modificare un singolo colore della ViewPort è disponibile la funzione:

```
SetRGB4( vp, c, rosso, verde, blu )
```

nella quale "vp" è un puntatore alla ViewPort, "c" è il numero del colore da modificare, e gli ultimi tre parametri sono i livelli dei colori fondamentali da utilizzare. La funzione:

```
SetRGB4CM( cm, c, rosso, verde, blu )
```

è del tutto analoga alla precedente, solo che accetta un puntatore alla struttura ColorMap ("cm") invece che alla ViewPort. Per leggere i livelli RGB di un colore è opportuno usare la funzione:

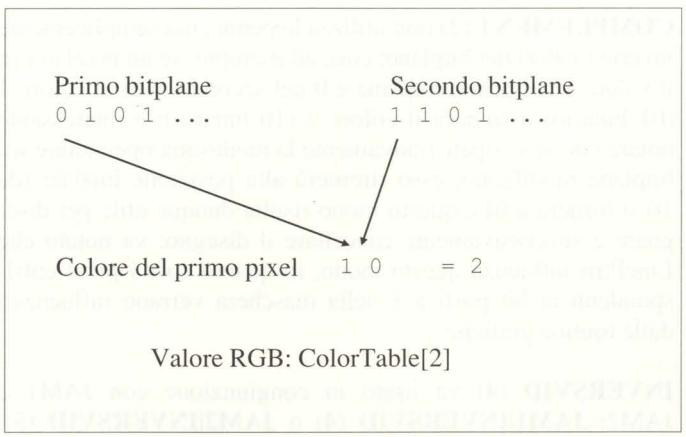
```
rgb = GetRGB4( cm, c )
```

la quale accetta un puntatore alla ColorMap e il numero "c" del colore da leggere; il valore di ritorno conterrà nei bit 8-11 il livello del rosso, nei bit 4-7 quello del verde e nei bit 0-3 quello del blu. Se ora sappiamo come vengono definiti e come si possono cambiare i colori di una ViewPort, rimane ancora da capire in che modo il sistema riesca ad associare a ogni pixel del video un determinato colore. La tecnica utilizzata implica l'impiego dei "bitplane" (piani di bit).

Bitplane

Nel nostro schermo d'esempio a 4 colori ogni pixel può assumere, come è ovvio, 4 differenti colori rappresentati dai numeri compresi tra 0 e 3. Per rappresentare tali numeri sono sufficienti, in numerazione binaria, 2 bit (0=00, 1=01, 2=10, 3=11); è evidente, quindi, che a ogni pixel dello schermo debbano corrispondere in memoria due bit. Tuttavia l'Amiga non conserva i due bit in locazioni di memoria consecutive, ma appunto attraverso i bitplane. La tecnica consiste nel collocare in posizioni adiacenti di memoria un solo bit per pixel; se, come nel

nostro caso, i bit necessari sono più di uno, il sistema porrà, a partire da un'altra locazione di memoria, un altro piano di bit contenente ancora un solo bit per pixel. Per stabilire, ad esempio, quale sia il colore del primo pixel, si dovrà andare a leggere il valore del primo bit del primo bitplane e il primo bit del secondo bitplane, combinarli insieme per formare una cifra a due bit e interpretarla come un valore numerico che indica quale delle UWORD della ColorTable vadano associate al pixel in questione, come mostra lo schema che segue:



Se la ColorTable fosse quella definita più sopra, il valore RGB del primo pixel sarebbe 000F, che corrisponde a un livello massimo di blu, niente verde e niente rosso. Se i bitplane fossero di più, il numero del colore sarebbe formato da ulteriori bit che andrebbero aggiunti alla sinistra di quelli già trovati. Ogni ViewPort può avere fino a 5 bitplane e quindi un massimo di 32 colori (un sesto bitplane viene utilizzato dal sistema in modalità HALF-BRITE per raggiungere 64 colori e in modalità HAM per ottenerne 4096, usando però delle tecniche in parte differenti). L'insieme dei bitplane associati alla ViewPort è detto "raster" e i singoli bitplane sono anche chiamati "piani di raster". La posizione in memoria dei bitplane è definita dai puntatori contenuti nella struttura BitMap, la quale contiene anche le informazioni relative al numero di bitplane impiegati, al numero di righe, al numero di colonne espresse in byte (e non in bit): ogni bitplane va infatti concepito come un rettangolo dotato di larghezza e altezza; va notato come ogni bitplane debba risiedere in chip ram (la memoria utilizzabile dai coprocessori grafici di Amiga, compresa, per ora, nei primi 512k della mappa di memoria) e iniziare a indirizzo pari; inoltre, deve avere un numero pari di byte per riga: così una ViewPort larga 7 pixel avrà un bitplane di due byte (16 bit) anche se il sistema ne riterrà significativi solo 7. Esiste una macro, definita in "gfx.h", che permette di ricavare il numero di byte complessivamente impiegati da un bitplane:

```
lunghezza = RASSIZE( largh, altez )
```

ove "largh" indica il numero di pixel sull'asse delle x, e "altez" quello sull'asse delle y, "lunghezza" esprime, appunto, il numero di byte necessari. La funzione:

```
bitplane = AllocRaster( largh, altez )
```

alloca memoria chip per un bitplane della larghezza e dell'altezza indicati, ritornando un puntatore al primo byte in caso di successso e 0 in caso di fallimento. La memoria per un bitplane può comunque essere allocata anche con le usuali funzioni Exec come AllocMem, avendo però l'accortezza di segnalare (con il flag MEMF_CHIP) che si desidera unicamente chip ram, e quella di calcolare con la macro RASSIZE la quantità di byte desiderati. Va sottolineato che una BitMap può essere più estesa dell'area visualizzata dalla ViewPort (fino a 1024 * 1024 pixel); quest'ultima, quindi, può essere concepita come una sorta di finestra aperta sulla BitMap, capace, fra altro, di scorrere in ogni direzione su di essa, visualizzandone man mano nuove parti; ciò si ottiene modificando le variabili contenute nella struttura RasInfo. Va ricordato che in modalità DUALPLAYFIELD (per intenderci quella tipica dei simulatori di volo, con un disegno di primo piano nei cui "buchi" appaiono i dettagli di un'immagine retrostante), la ViewPort ha associate due BitMap, una per l'immagine superiore e una per quella inferiore; infine, viene spesso usata una tecnica di "double buffering", in cui, mentre il sistema visualizza una BitMap, il programma utente lavora su una seconda, che, una volta modificata, verrà passata al sistema per la visualizzazione, evitando così lo sfarfallio generato dai cambiamenti a vista.

Disegnare sull'Amiga non significa altro che andare a modificare i bitplane per mutare il colore di uno o più pixel. Questo può essere fatto direttamente, ma risulta, ovviamente, piuttosto complicato. Per aiutare l'utente, la libreria grafica mette a disposizione una serie di funzioni espressamente dedicate al disegno, che fanno tutte riferimento a una particolare struttura: la RastPort.

RastPort

La RastPort contiene tutte le informazioni che permettono di pilotare le routine grafiche relative al disegno, all'animazione, alla scrittura dei caratteri di testo. Essa può essere direttamente inizializzata dall'utente, cosa che implica una più profonda conoscenza delle View e delle ViewPort, oppure può essere condivisa con Intuition, traendone l'indirizzo o dalla struttura Window:

```
rp = w->RPort;
```

o dallo Screen:

```
rp = &s->RastPort;
```

"rp" è il puntatore alla RastPort, "w" il puntatore alla Window, "s" il puntatore allo Screen.

Si noti che le due RastPort sono differenti; che esse vengono utilizzate anche da Intuition per disegnare i bordi e i gadget di sistema; che Intuition fa uso della "layers.library" per gestire le finestre e che quindi bisognerebbe conoscerne i meccanismi di funzionamento per evitare effetti non previsti; che, fra le due, è meglio utilizzare la RastPort dello Screen, anche se questo potrà modificare il modo in cui vengono disegnati i Menu, oppure conviene ricorrere a una Window del tipo GIMMEZEROZERO. Ovviamente la RastPort dello Screen ci permetterà di disegnare in tutto lo Screen: le coordinate 0,0 si

riferiranno al suo punto in alto a sinistra e verranno incrementate muovendosi verso destra e verso il basso.

Le penne

La maggior parte delle routine grafiche non permette di indicare direttamente il colore da utilizzare per disegnare, esse invece fanno riferimento a delle variabili contenute nella RastPort, note come "drawing pens" (penne da disegno). La prima è detta "APen" o "FgPen" ("Foreground Pen") e il suo colore viene stabilito dalla funzione:

```
SetAPen( rp, c )
```

ove "rp" è il puntatore alla RastPort e "c" è il numero del colore da utilizzare, d'ora in poi, con questa penna. La seconda è detta "BPen" o "BgPen" ("Background Pen") e il colore lo si imposta con la funzione:

```
SetBPen( rp, c )
```

i cui parametri sono analoghi a quella precedente. Una terza penna, chiamata OPen, riguarda il disegno delle aree, per cui ne parleremo al momento opportuno. L'uso delle penne da parte delle routine di disegno viene influenzato dalle maschere e dal modo grafico.

Le maschere

Le maschere sono tre: per le linee, per le aree, per i bitplane. L'USHORT (unsigned short) "RastPort.LinePtrn" ("Line Pattern") è una maschera che entra in gioco nel disegno delle linee; va concepito come un esempio della linea da disegnare lungo 16 pixel, corrispondenti ai 16 bit dell' USHORT. Ogni bit posto a 1 indica al sistema di utilizzare la APen per disegnare quel pixel, ogni 0 indica di lasciarlo immutato o di usare la BPen (a seconda del modo grafico). Per esempio, se LinePtrn è 0xFFFF (1111111111111111 binario), ogni linea verrà tracciata come linea piena; se, invece, fosse 0xF0F0 (in binario 1111000011110000), la linea apparirebbe come una serie di trattini. Per modificare tale maschera occorre usare la macro:

```
SetDrPt( rp, maschera )
```

ove "maschera" è l'USHORT che abbiamo scelto. La maschera per le aree, "RastPort.AreaPtrn", verrà esaminata meglio più oltre, ma il suo significato è analogo a quello di LinePtrn, solo che si riferisce al modo in cui devono essere riempite le aree. La maschera per i bitplane, "RastPort.Mask", è un UBYTE (unsigned char) i cui bit posti a 1 indicano i bitplane che possono essere modificati dalle routine grafiche; quelli pari a 0, saranno lasciati invariati.

Il modo grafico

È indicato dal valore di "RastPort.DrawMode" e può essere modificato dalla funzione:

```
SetDrMd( rp, modo )
```

Il parametro "modo" può assumere i seguenti valori:

JAM1 (0) prevede che sia utilizzata solo la APen, così i bit posti a 0 delle maschere non verranno modificati quando si disegna;

JAM2 (1) utilizza entrambe le penne; i pixel corrispondenti ai bit posti a 1 delle maschere assumeranno il colore della APen, quelli pari a 0 il colore della BPen;

COMPLEMENT (2) non utilizza le penne, ma semplicemente inverte i valori dei bitplane; così, ad esempio, se un pixel aveva il valore 1 nel primo bitplane e 0 nel secondo, pari al colore 1 (01 binario) assumerà il colore 2 (10 binario); è interessante notare che se si ripete nuovamente la medesima operazione sul bitplane modificato, esso ritornerà alla posizione iniziale (da 10 si tornerà a 01); questo modo risulta dunque utile per disegnare e successivamente cancellare il disegno; va notato che LinePtrn influenza questo modo, in quanto solo i pixel corrispondenti ai bit posti a 1 nella maschera verrano influenzati dalle routine grafiche.

INVERSVID (4) va usato in congiunzione con JAM1 e JAM2: **JAM1|INVERSVID (4)** o **JAM2|INVERSVID (5)**; viene utilizzato generalmente con i testi e permette nel primo caso di stampare il carattere utilizzando la APen nell'area che circonda il carattere, lasciando inalterato il colore dei restanti pixel. Con JAM2, invece, lo sfondo appare sempre in APen, mentre il carattere vero e proprio appare in BPen.

Il trattamento dei pixel

Due funzioni permettono di scrivere e leggere singoli pixel:

```
errore = WritePixel( rp, x, y )
```

usa il colore della APen per modificare il pixel posto nella locazione "x", "y". Ritorna -1 se le coordinate cadono al di fuori dei confini della BitMap.

```
c = ReadPixel( rp, x, y )
```

ritorna il numero del colore del pixel posto alla posizione definita da "x" e "y". Se il valore è pari a -1 il punto si trova al di fuori della BitMap. Va notato che queste sono le sole funzioni che controllano i parametri che si passano loro per stabilire se sono stati rispettati i limiti della BitMap, tutte le altre eseguiranno comunque l'operazione richiesta, con alte possibilità di corrompere il sistema, se i valori passati risultano errati.

Il trattamento delle linee

Per tracciare una linea occorre dapprima stabilirne il punto di partenza tramite la funzione:

```
Move( rp, x, y )
```

che rende il punto "x", "y" posizione corrente della RastPort; poi definire il punto di arrivo con la funzione:

```
Draw( rp, x, y )
```

questa funzione oltre a disegnare la linea, rende il punto di arrivo la nuova posizione corrente. La funzione:

```
PolyDraw( rp, n, vrt )
```

permette di disegnare una linea spezzata di "n" vertici, le cui coordinate vanno poste in un array di SHORT, indirizzato da "vrt", che contenga due elementi per ogni vertice; ad esempio:

```
vrt[] = { 10,20, 100,50, 40,30 };
PolyDraw( rp, 3, &vrt[0] );
```

disegnerà una linea che parte dalla posizione corrente passa attraverso i punti 10,20 e 100,50 e termina al punto 40,30, che diventa la nuova posizione corrente. Tutte queste funzioni rispettano a pieno titolo il modo grafico e le maschere correnti. Per disegnare ellissi è disponibile la funzione:

```
DrawEllipse( rp, cx, cy, o, v )
```

la quale disegna un ellisse di centro "cx", "cy", raggio verticale "v" e orizzontale "o". La funzione si distingue dalle precedenti perché non fa uso della maschera delle linee e quindi disegna sempre linee piene con la APen sia in JAM1 che in JAM2; rispetta comunque il modo COMPLEMENT. Inoltre non influisce sulla posizione corrente e nemmeno ne è influenzata. La macro:

```
DrawCircle( rp, cx, cy, r )
```

non fa altro che chiamare DrawEllipse con raggio verticale e orizzontale uguali.

Le aree: la OPen e le maschere

Per area si intende, in generale, una superficie interamente colorata di forma varia. Le routine che le gestiscono, oltre a fare riferimento alle variabili della RastPort già citate, sono direttamente influenzate dal valore della "OPen" o "AOlPen" ("Area Outline Pen"), essa indica il colore della linea che circonderà l'area da colorare; la macro:

```
SetOPen( rp, c )
```

rende "c" il colore di questa penna e aggiunge al campo "RastPort.Flag" il flag AREAOUTLINE (8) il quale indica, appunto, che le aree piene vanno bordate con una linea di questo colore; va subito precisato che la linea del bordo viene sempre e comunque disegnata in modo JAM1 e, quindi, non viene mai usata la BPen o il complemento dei colori; il valore di "RastPort.Mask" incide su tale penna come sulle altre penne; invece, la maschera contenuta in LinePtrn viene utilizzata solo in un caso, come vedremo; infine, l'uso della OPen modifica la posizione corrente per il tracciamento delle linee. Per disabilitare la OPen occorre usare la macro:

```
BNDRYOFF( rp )
```

la quale, semplicemente, cancella il flag AREAOUTLINE. Le aree hanno una loro maschera indirizzata dalla variabile "RastPort.AreaPtrn"; tale maschera deve essere costituita da un array di UWORD che viene interpretato dal sistema come la descrizione di un'area rettangolare larga sempre 16 pixel (i 16 bit della UWORD) e alta un numero variabile di pixel; questo numero deve essere una potenza di 2 e può quindi assumere i valori 2 4 8 16 e così via; ovviamente, se l'altezza fosse 8, 8 devono essere le UWORD contenute nell'array; la macro che permette di modificare questa maschera è:

```
SetAfPt( rp, maschera, e )
```

ove "maschera" è l'indirizzo dell'array di UWORD ed "e" rappresenta l'esponente di 2 che esprime l'altezza della maschera: così se l'altezza fosse 8, "e" dovrebbe assumere il valore 3 (2 elevato 3 = 8); in altre parole, è il logaritmo in base 2 dell'altezza.

La maschera per le aree ha il medesimo significato di quella per le linee e va interpretata in combinazione con il modo grafico. Tuttavia, essa ha un'origine fissa: il punto 0,0 della RastPort; se l'area disegnata, per esempio, fosse un rettangolo con le seguenti coordinate: 4,0, 8,0 8,30 30,4, la parte della maschera compresa tra il primo e il quarto bit di ogni singola UWORD non verrebbe utilizzata. Esiste un modo diverso di utilizzare la maschera per le aree, che viene chiamato "multicolore"; in questo caso occorre definire un numero di maschere esattamente pari ai bitplane della BitMap cui è collegata la RastPort e porli uno di seguito all'altro in un array di UWORD. Per uno schermo di due bitplane, una maschera multicolore alta 8 pixel sarà costituita da 16 UWORD che il sistema interpreterà come un vero e proprio raster largo 16 pixel, da sostituire a quello esistente nella BitMap, laddove andremo a disegnare aree piene. Per abilitare la maschera multicolore si usa ancora SetAfPt, indicando però con un numero negativo il parametro "e"; nel nostro esempio, dovrà assumere il valore -3 perché l'altezza della maschera è di 8 pixel. Per disabilitare la maschera e disegnare così aree piene solide, occorre chiamare SetAfPt con il parametro "maschera" uguale a 0.

La costruzione di aree

Un primo modo per disegnare aree piene consiste nel costruirle in un buffer temporaneo e una volta concluse disegnarle sullo schermo. Per poter usare le routine di questo tipo occorre modificare due variabili della RastPort: AreaInfo e TmpRas. La preparazione di queste due variabili è piuttosto lunga e implica le seguenti operazioni:

a) preparare un buffer per contenere le coordinate dei vertici dell'area che disegneremo; questo buffer deve essere allocato dall'utente, posto nella chip ram a partire da un indirizzo pari e contenere 5 byte per ogni vertice (anche se pare che il sistema ne usi solo 4);

b) inizializzare una struttura AreaInfo con la funzione:

```
InitArea( ai, buffer, n )
```

in cui "ai" punta alla struttura AreaInfo, "buffer" punta alla chip ram e "n" indica il numero di vertici della nostra area;

c) porre l'indirizzo "ai" in "RastPort.AreaInfo";

d) creare un bitplane che verrà usato dalle funzioni della libreria per tracciare l'area prima di riversarla sullo schermo. Il bitplane deve essere abbastanza grande da contenere la nostra area e conviene, in generale, farlo tanto ampio quanto la BitMap da noi usata. Per allocarne la memoria si può chiamare la funzione AllocRaster;

e) inizializzare una struttura TmpRas con la funzione:

```
tmpras = InitTmpRas( tr, bitplane, lungh )
```

"tr" deve puntare alla struttura TmpRas, "bitplane" al bitplane temporaneo creato con AllocRaster, mentre "lungh" deve contenere la lunghezza in byte del bitplane, che si ottiene con la macro RASSIZE;

f) Porre il valore di ritorno della funzione precedente ("tmpras") in "RastPort.TmpRas". A questo punto la RastPort è pronta a ricevere i comandi per la costruzione di aree piene. Molte delle funzioni che ora illustreremo restituiscono un errore (-1) quando il buffer di AreaInfo non è sufficiente per contenere i vertici che intendiamo disegnarvi; tutte, inoltre, usano la maschera per le aree piene, se è stata definita. Il disegno di aree poligonali piene deve essere iniziato con la funzione:

```
errore = AreaMove( rp, x, y )
```

che aggiunge un vertice al buffer e dà inizio a una nuova area; se esisteva un'area già aperta, essa viene chiusa, ma non viene tracciata sullo schermo; l'area nuova verrà aggiunta alla precedente, e le zone di intersezione saranno considerate vuote; in generale vale la regola: se un numero pari di aree si sovrappone l'area è considerata vuota, viceversa se il numero è dispari, l'area viene considerata piena al momento della visualizzazione (lo si può considerare un XOR logico fra le aree). Con la funzione:

```
errore = AreaDraw( rp, x, y )
```

si aggiunge un nuovo vertice al buffer, congiunto al precedente da un segmento che rappresenta un lato dell'area da riempire. Con

```
errore = AreaEnd( rp )
```

si aggiunge un nuovo vertice al buffer, le cui coordinate saranno uguali a quelle dell'ultima chiamata ad AreaMove, chiudendo così l'area; è evidente, allora, che per disegnare un triangolo occorrono 4 vertici, per un rettangolo 5 e così via. L'area viene visualizzata sullo schermo e il buffer viene azzerato, per cui le prossime istruzioni di AreaMove e AreaDraw avranno a disposizione tutti i byte del buffer e le nuove aree disegnate non andranno a intersecarsi con le precedenti. Se la OPen è abilitata, l'area verrà circondata da una linea nel colore della OPen, qualunque sia il modo grafico, e la linea sarà comunque piena (cioè non verrà rispettato il valore di LinePtrn). La funzione:

```
errore = AreaEllipse( rp, cx, cy, o, v )
```

aggiunge due vertici al buffer, preparando i valori per il disegno di un ellisse di centro "cx", "cy", raggio verticale "v" e orizzontale "o". L'area ellittica verrà visualizzata al prossimo AreaEnd e potrà intersecarsi con altre aree come avviene per le aree poligonali. Tuttavia, per evitare strani effetti, conviene disegnare prima i poligoni, poi le ellissi e subito dopo chiamare AreaEnd. Le ellissi non supportano la OPen. La macro:

```
errore = AreaCircle( rp, cx, cy, r )
```

non fa altro che chiamare AreaEllipse con raggio orizzontale e verticale uguali, disegnando così un cerchio. L'ultima funzione che permette di costruire aree piene è

```
RectFill( rp, x, y, xmax, ymax )
```

essa disegna a notevole velocità e accetta come parametri le coordinate degli angoli in alto a sinistra e in basso a destra del rettangolo da disegnare. La funzione usa la maschera per le aree e la OPen, che, in questo caso, rispetta anche il LinePtrn; inoltre non richiede che AreaInfo o TmpRas siano inizializzate: da questo punto di vista appare la funzione più efficiente per la costruzione delle aree.

Il riempimento delle aree

La libreria grafica presenta anche una funzione che permette di riempire delle aree già in qualche modo definite sullo schermo. Si tratta di

```
errore = Flood( rp, modo, x, y )
```

Il suo funzionamento dipende strettamente dal parametro "modo", che puo assumere due valori (0 o 1). Quando è uguale a 0 si richiede il cosiddetto modo "outline": in questo caso la funzione partirà dalle coordinate "x","y" e, usando le penne e il modo grafico corrente, colorerà tutti i pixel adiacenti il cui colore è diverso da quello contenuto nella OPen. Non vanno considerati adiacenti i pixel che si toccano solo per un angolo. Se l'area non è completamente circondata dal colore della OPen, tutto il raster verrà riempito da Flood.

Se il parametro "modo" è uguale a 1, si richiede il modo "color": il punto "x", "y", allora, non stabilisce solo le coordinate da cui partire per il riempimento, ma anche il colore da modificare: saranno mutati, infatti, tutti i pixel adiacenti del suo stesso colore, utilizzando il modo grafico corrente. La funzione Flood usa la maschera per le aree, anche in modo multicolore; rispetta la maschera dei bitplane; non richiede una struttura AreaInfo. Per quanto riguarda TmpRas, il discorso si fà più complesso: in primo luogo va detto che la documentazione ufficiale sembra ritenere necessario inizializzare TmpRas; di fatto, quando ciò viene fatto, la funzione Flood appare molto più veloce;

continua a pag 70

L'uso delle librerie da Basic

Le librerie "shared" (condivise) sono una collezione di routine che i vari task del sistema possono utilizzare contemporaneamente, garantendo così un notevole risparmio di memoria. Esse inoltre offrono un controllo sulle potenzialità della macchina che è quasi totale. L'AmigaBasic permette di accedervi con notevole facilità: basta aprirle con il comando

```
LIBRARY "nome_della_libreria.library"
```

i nomi delle librerie devono sempre terminare con l'estensione ".library". Perché una libreria possa essere aperta dall'interprete, occorre che esso abbia a disposizione un file con lo stesso nome della libreria, ma terminante con l'estensione ".bmap" (basic map). Tale file contiene informazioni sui parametri da passare alla funzione e sulla sua posizione relativa rispetto alla base della libreria (offset). Questi file non sono forniti assieme al Basic, se si fa eccezione delle librerie Exec, Dos e Graphics i cui file bmap si trovano nella directory basicdemos del disco Extras; i file mancanti possono essere facilmente creati dall'utente usando il programma Basic ConvertFD, sempre nella directory basicdemos. Tale programma legge i file del tipo "nome_della libreria.fd" contenuti nella directory fd1.3 (o fd1.2 per la versione 1.2 del sistema operativo), sempre del disco Extras e li traduce in file bmap. I file fd contengono praticamente le stesse informazioni dei file bmap, ma in un formato leggibile direttamente da un essere umano; vengono forniti dalla Commodore, aggiornati a ogni nuova versione delle librerie ed è quindi sempre utile consultarli. Il programma si preoccupa anche di aggiungere una "x" minuscola iniziale a quelle routine il cui nome risulta in conflitto con le parole riservate del Basic (come la funzione Dos "Open" che diventa "xOpen"): se ne veda l'elenco nel listato del programma ConvertFD. Perché l'interprete Basic possa individuare il file bmap, esso deve essere posto o nella directory corrente del programma Basic, o nella directory "LIBS:" o ancora nella directory che precede il nome della libreria nel comando LIBRARY; in questo esempio

```
LIBRARY "df1:prova/graphics.library"
```

il Basic cercherà il file "graphics.bmap" (non il file "graphics.library") nella directory "df1:prova". Se una funzione ritorna un valore occorre dire al Basic di che tipo di valore si tratta; per fare questo si usa il comando:

```
DECLARE FUNCTION nome_funzione LIBRARY
```

il nome della funzione non va posto fra virgolette, ma va invece seguito da un'estensione che dichiari il valore ritornato. Questo è generalmente un LONG e quindi l'estensione sarà costituita dal carattere "&". La funzione può essere quindi usata con la sintassi:

```
CALL nome_funzione( par_1, ... par_n)
```

oppure, se ritorna un valore:

```
ritorno = nome_funzione( par_1, ... par_n)
```

Il nome della funzione è sensibile al maiuscolo e al minuscolo, pertanto occorre rispettare sempre esattamente tali caratteristiche.

I parametri da passare alle funzioni sono o puntatori o LONG, pertanto dovranno avere anch'essi l'estensione "&" (in Basic, di default le variabili sono numeri in virgola mobile). Per evitare di dover appesantire il listato (e le dita) con lo scomodo carattere "&" conviene cambiare il valore di default di tutte le variabili con il comando:

```
DEFLNG a-z
```

come abbiamo fatto nel programma d'esempio. Per passare una costante o una variabile alla funzione chiamata, basta indicarla come parametro, facendo attenzione al fatto che, in Basic, non esistono valori senza segno e che quindi occorre tradurre gli short superiori a 32767 in valori negativi, sottraendogli 65536 (ad es. 32768-65536 = -32678). Per passare l'indirizzo di una variabile, occorre usare la funzione:

```
VARPTR( variabile )
```

per un array:

```
VARPTR( array(0) )
```

cioè l'indirizzo del primo elemento. L'indirizzo di una stringa, la quale generalmente deve terminare con uno 0, va passato con

```
SADDR( stringa$ + CHR$(0) )
```

Le strutture possono essere poste in array, inizializzate attraverso un ciclo READ-DATA, per poi passare l'indirizzo del primo elemento alla funzione. Questo metodo va, a nostro avviso, del tutto evitato, perché le variabili del Basic non rimangono fisse in memoria e quindi il loro indirizzo varia con lo svolgersi del programma in Basic, quando vengono aggiunte nuove variabili o vengono chiamate subroutine. Pertanto è meglio allocare memoria con la funzione Exec AllocMem (la cui libreria, a differenza del C, va aperta per mettere a disposizione il relativo file bmap all'interprete); mediante essa, inoltre, è possibile allocare memoria in chip ram, cosa impossibile da Basic e necessaria viceversa per alcune routine grafiche.

inoltre il progressivo riempimento dell'area non appare sullo schermo, ma avviene sul raster temporaneo; così, sullo schermo comparirà, di colpo, il risultato finale. TmpRas, inoltre, appare sempre necessario in modo "color". In modo "outline", sembra indispensabile solo quando si usa la maschera delle aree; se la maschera è multicolore e TmpRas non appare implementato, la funzione non ritorna più e occorre resettare la macchina (può giungere anche un guru); se invece la maschera è normale, in modo "color" ritorna un errore (0), in modo "outline" si comporta come se la maschera non esistesse. Lo specchietto che segue, riassume il comportamento di Flood, quando TmpRas non è stata inizializzata:

	maschera delle aree		
	senza	normale	multicolore
"outline"	va bene	non la usa	non ritorna
"color"	errore (0)	errore (0)	non ritorna

Appare dunque conveniente preparare TmpRas per l'uso di Flood, a meno che non si desideri ottenere, a nostro rischio e pericolo, l'effetto del riempimento a vista in modo "outline". Infine, citiamo un'ultima funzione da noi utilizzata nel programma esempio: si tratta di

```
SetRast( rp, c )
```

che permette di riempire l'intero raster con il colore specificato ("c"), cancellando qualsiasi cosa esso contenga.

I programmi

Si tratta di due programmi quasi identici scritti in C e in Basic, con intenti puramente didascalici: essi permettono di vedere all'opera un buon numero di funzioni e parametri grafici, per rendersi conto, dal vivo, di come lavorano le routine citate. Il lettore dovrebbe utilizzarli modificando parametri e funzioni per approfondire la conoscenza del sistema. Il programma in C non visualizza i valori della RastPort, a differenza di quello in Basic, per evitare un lungo elenco di IntuiText che avrebbe reso il listato eccesivamente ingombrante. Il programma in Basic dimostra, in particolare, l'uso delle librerie di sistema (riferirsi al contenuto di pagina xx per una trattazione più completa dell'utilizzo delle librerie di sistema da Basic) e può essere utile per testare le funzioni senza dovere ogni volta compilare e linkare un sorgente C.

Programma in Basic

```
REM Test Basic
DEFLNG a-z

DEF FNRASSIZE(largh,altez) = altez*(((largh+15)/8)
AND -2)
DECLARE FUNCTION AllocMem LIBRARY
DECLARE FUNCTION AllocRaster LIBRARY
DECLARE FUNCTION InitTmpRas LIBRARY
LIBRARY "df1:basicdemos/graphics.library"
LIBRARY "df1:basicdemos/exec.library"

DIM lineamask%(16), rgb%(32), modi$(2)

modi$(0)=" JAM1": modi$(1)=" JAM2"

REM variabili azzerate per fine
bitplane = 0: areabuffer = 0: dati = 0
prof = 5: largh = 320: altez = 200

SCREEN 2,largh,altez,prof,1
WINDOW 2,"Click tasto sinistro",,7,2

pWindow   = WINDOW(7)
pScreen   = PEEKL(pWindow + 46)
ViewPort  = pScreen + 44

REM Usiamo la RastPort dello Screen
rp        = pScreen + 84

REM La memoria che ci serve per le strutture
REM indirizzata dalla variabile "dati"
REM                dati+    lunghezza
REM vertici          0        32  byte
REM TmpRas          32         8  byte
REM AreaInfo        40        32  byte
REM areamask        72        16  byte
REM areamaskcol     88        80  byte
REM                TOT       168  byte
lunghezza = 168
dati = AllocMem(lunghezza,CLNG(2^16+1))
IF dati = 0 THEN fine
drawvertici = dati        'per PolyDraw()
TmpRas      = dati + 32   'per InitTmpRas()
AreaInfo    = dati + 40   'per InitAreaInfo()
areamask    = dati + 72   'per SetAfPt()
areamaskcol = dati + 88   'per SetAfPt()

REM Inizializziamo la memoria allocata
CALL inizmem(drawvertici,16)
DATA 10,20, 100,20, 100,80, 40,80
DATA 40,35,  80,35,  80,65, 10,65
CALL inizmem(areamask,8)
DATA &HFF00, &HFF00, &HFF00, &HFF00
DATA &H00FF, &H00FF, &H00FF, &H00FF
CALL inizmem(areamaskcol,40)
DATA &HFFFF, &HFFFF, &HFFFF, &HFFFF
DATA &H0000 &H0000, &H0000, &H0000
DATA &HFF00, &HFF00, &HFF00, &HFF00
DATA &H00FF, &H00FF, &H00FF, &H00FF
DATA &H0FF0, &H0FF0, &H0FF0, &H0FF0
DATA &H0FF0, &H0FF0, &H0FF0, &H0FF0
DATA &H00FF, &H00FF, &H00FF, &H00FF
DATA &HFF00, &HFF00, &HFF00, &HFF00
DATA &H0000, &H0000, &H0000, &H0000
DATA &H0000, &H0000, &H0000, &H0000
```

```
REM Prepariamo AreaInfo
numareavt = 9
areabuffer = AllocMem(numareavt*5,CLNG(2^16+2+1))
IF areabuffer = 0 THEN fine
CALL InitArea(AreaInfo,areabuffer,numareavt)
POKEL rp+16,AreaInfo

REM Prepariamo TmpRas
bitplane = AllocRaster(largh,altez)
IF bitplane = 0 THEN fine
tr=InitTmpRas(TmpRas,bitplane,FNRASSIZE(largh,altez))

POKEL rp+12,tr

REM Prepariamo i valori per LinePtrn
FOR i = 0 TO 15
  READ lineamask%(i)
NEXT
DATA &HFFFF,&HF0F0,&HFF00,&HF000
DATA &HCCCC,&HC0C0,&HCC00,&HC000
DATA &HCFCF,&HFCFC,&HFFCC,&HCCFF
DATA &H8888,&H8080,&H8800,&HF18F

REM Prepariamo la tavola dei colori...
rgb%(0)=&H444: rgb%(1)=&HAAA
FOR i=2 TO 31
   rgb%(i)=RND*&HFFF
NEXT
REM  ...la carichiamo...
CALL LoadRGB4(ViewPort,VARPTR(rgb%(0)),32)
REM  ...e la visualizziamo
FOR i = 0 TO 31
   LOCATE 12,i+1: COLOR i: PRINT "*"
   LOCATE 13,i+1: COLOR 1: PRINT USING "#";i MOD 10
   IF i MOD 10 = 0 THEN LOCATE 14,i+1: COLOR 1:
PRINT USING "#";i/10
NEXT

ON BREAK GOSUB fine
BREAK ON
LOCATE 16,22: PRINT "JAM1      JAM2"

REM Stabiliamo il modo grafico
FOR modografico = 0 TO 1
   CALL SetDrMd(rp,modografico)

   FOR z=2 TO 15
      REM  Stabiliamo LinePtrn, FgPen, BgPen, AOlPen
      REM  e AreaPtrn
      CALL SetDrPt(rp,lineamask%(z))
      CALL SetAPen(rp,z)
      CALL SetBPen(rp,z-1)
      CALL SetOPen(rp,z+1)
      CALL SetAfPt(rp,areamask,3)
      REM  Disegnamo una linea ...
      CALL Move(rp,160+modografico*70,140+z*3)
      CALL Draw(rp,220+modografico*70,140+z*3)
      REM  ...e una poligonale
      CALL Move(rp,10,65)
      CALL PolyDraw(rp,8,drawvertici)
      REM  Costruiamo un area quadrata e
      REM  due circolari
      CALL AreaMove(rp,150,20)
      CALL AreaDraw(rp,210,20)
      CALL AreaDraw(rp,210,80)
      CALL AreaDraw(rp,150,80)
      CALL AreaEllipse(rp,150,50,30,30)
      CALL AreaEllipse(rp,150,50,15,15)
      CALL AreaEnd(rp)
      REM  Costruiamo un rettangolo pieno
      CALL RectFill(rp,223,20,300,80)
      GOSUB printrp
      Attesa1: SLEEP: IF MOUSE(0) < 1 THEN Attesa1
   NEXT z
NEXT modografico

REM  Riempiamo lo Screen col colore 15
CALL SetRast(rp,15&)
REM  Disegnamo un cerchio...
CALL SetDrMd(rp,1)
CALL SetDrPt(rp,&HFFFF)
CALL SetAPen(rp,10)
CALL DrawEllipse(rp,150,100,90,90)
REM  ...e lo riempiamo con la maschera multicolore
CALL SetAfPt(rp,areamaskcol,&HFD)
CALL SetAPen(rp,&HFF)
CALL SetBPen(rp,0)
CALL SetOPen(rp,10&)
CALL Flood(rp,0,150,100)
LOCATE 11,11: PRINT "Flood multicolore"
Attesa2: SLEEP: IF MOUSE(0) < 1 THEN Attesa2

fine:
REM Liberiamo le risorse allocate
IF dati <> 0 THEN CALL FreeMem(dati,lunghezza)
IF areabuffer <> 0 THEN
           CALL FreeMem(areabuffer,numareavt*5)
IF bitplane <> 0 THEN
           CALL FreeRaster(bitplane,largh,altez)
SCREEN CLOSE 2
LIBRARY CLOSE
END

printrp:
REM Visualizza alcuni campi della RastPort
drawmode     = PEEK(rp+28)
FgPen        = PEEK(rp+25)
BgPen        = PEEK(rp+26)
AOlPen       = PEEK(rp+27)
LinePtrn     = PEEKW(rp+34)
LOCATE 16,1: PRINT "RastPort  ": PRINT
PRINT "DrawMode  "; modi$(drawmode)
CALL prcol("FgPen     ",FgPen)
CALL prcol("BgPen     ",BgPen)
CALL prcol("AOlPen    ",AOlPen)
PRINT "LinePtrn  "; " ";HEX$(CINT(LinePtrn))
RETURN

SUB inizmem(puntatore,num%) STATIC
FOR i = 0 TO num%-1
```

```
   READ v%: POKEW puntatore + i*2, v%
NEXT
END SUB

SUB prcol(a$,c) STATIC
PRINT a$; c;: COLOR c: PRINT TAB(15) "*": COLOR 1
END SUB

SUB SetDrPt(rp,ptrn%) STATIC
POKEW rp+34,ptrn%: POKE rp+30,15
POKEW rp+32,PEEKW(rp+32) OR 1
END SUB

SUB SetAfPt(rp,ap,n%) STATIC
POKEL rp+8,ap: POKE rp+29,n%
END SUB

SUB SetOPen(rp,c) STATIC
POKE rp+27,c: POKEW rp+32,PEEKW(rp+32) OR 8
END SUB
```

Listato in C

```
/* Test C
compilare con lc -Lcd test.c ( Lattice )
*/
#include "exec/types.h"
#include "exec/memory.h"
#include "hardware/custom.h"
#include "graphics/gfx.h"
#include "graphics/view.h"
#include "graphics/rastport.h"
#include "graphics/gfxmacros.h"
#include "intuition/intuition.h"
#include "proto/exec.h"
#include "proto/graphics.h"
#include "proto/intuition.h"

#define numareavt 9
#define largh 320
#define altez 200
#define prof 5

int Attesa();

struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct NewScreen ns =
    {
    0,0,
    largh,altez,prof,
    2,1,
    0,
    CUSTOMSCREEN,
    0,
    0,
    0,
    0
    };
struct NewWindow nw =
    {
    0,0,
    largh,altez,
    2,1,
    CLOSEWINDOW|MOUSEBUTTONS,
    WINDOWSIZING|WINDOWCLOSE|NOCAREREFRESH|
    ACTIVATE|SIMPLE_REFRESH,
    NULL, NULL,
    "Click tasto sinistro",
    NULL, NULL,
    150,20,320,200,
    CUSTOMSCREEN
    };
struct Screen *s;
struct Window *w;
struct IntuiMessage *im;
struct RastPort *rp;
struct ViewPort *vp;
struct AreaInfo areainfo, *ai;
struct TmpRas tmpras,*tr;
short * areabuffer;
PLANEPTR bitplane;
UWORD drawvertici[] = {
    10,20, 100,20, 100,80, 40,80,
    40,35,  80,35,  80,65, 10,65
    };
UWORD areamask[] = {
    0xFF00, 0xFF00, 0xFF00, 0xFF00,
    0x00FF, 0x00FF, 0x00FF, 0x00FF
    };
UWORD areamaskcol[] = {
    0xFFFF, 0xFFFF, 0xFFFF, 0xFFFF,
    0x0000, 0x0000, 0x0000, 0x0000,
    0xFF00, 0xFF00, 0xFF00, 0xFF00,
    0x00FF, 0x00FF, 0x00FF, 0x00FF,
    0x0FF0, 0x0FF0, 0x0FF0, 0x0FF0,
    0x0FF0, 0x0FF0, 0x0FF0, 0x0FF0,
    0x00FF, 0x00FF, 0x00FF, 0x00FF,
    0xFF00, 0xFF00, 0xFF00, 0xFF00,
    0x0000, 0x0000, 0x0000, 0x0000,
    0x0000, 0x0000, 0x0000, 0x0000
    };
UWORD lineamask[] = {
    0xFFFF,0xF0F0,0xFF00,0xF000,0xCCCC,0xC0C0,
    0xCC00,0xC000,0xCFCF,0xFCFC,0xFFCC,0xCCFF,
    0x8888,0x8080,0x8800,0xF18F
    };
UWORD rgb[] = {
    0x0444,0x0ccc,0x0c00,0x00c0,0x000c,0x0cc0,
    0x00cc,0x0c0c,0x0a4a,0x04aa,0x0aa4,0x0a82,
    0x028a,0x02a8,0x08a2,0x0a28,0x082a,0x008a,
    0x00a8,0x0a08,0x080a,0x08a0,0x0a80,0x0b70,
    0x007b,0x07b0,0x070b,0x0b70,0x00b7,0x0999,
    0x0909,0x0990
    };

void main()
```

```
{
short modografico,z;
IntuitionBase = (struct IntuitionBase *)OpenLibrary
                        ("intuition.library",0L);
if (IntuitionBase == NULL) goto fine;
GfxBase = (struct GfxBase *)OpenLibrary
                         ("graphics.library",0L);
if (GfxBase == NULL) goto fine;
s = OpenScreen(&ns);
if (s == NULL) goto fine;
nw.Screen = s;
w = OpenWindow(&nw);
if(w == NULL) goto fine;
vp = &s->ViewPort;
/* Usiamo la RastPort dello Screen */
rp = &s->RastPort;
ai=&areainfo;
tr=&tmpras;
/* Prepariamo AreaInfo */
areabuffer = (short *)
       AllocMem(numareavt*5,MEMF_CLEAR|MEMF_CHIP);
if (areabuffer == NULL) goto fine;
InitArea(ai,areabuffer,numareavt);
rp->AreaInfo=ai;
/* Prepariamo TmpRas*/
bitplane = AllocRaster(largh,altez);
if (bitplane == NULL) goto fine;
rp->TmpRas =
     InitTmpRas(tr,bitplane,RASSIZE(largh,altez));

/* Carichiamo la tavola dei colori*/
LoadRGB4(vp,&rgb[0],32);
for(modografico = 0; modografico < 2; modografico++)
    {
    /*Stabiliamo il modo grafico*/
    SetDrMd(rp,modografico);
    for(z=2;z<15;z++)
        {
        /* Stabiliamo LinePtrn, FgPen, BgPen, AOlPen
           e AreaPtrn */
        SetDrPt(rp,lineamask[z]);
        SetAPen(rp,z);
        SetBPen(rp,z-1);
        SetOPen(rp,z+1);
        SetAfPt(rp,&areamask[0],3);

    /*  Disegnamo una linea ...*/
        Move(rp,160+modografico*70,140+z*3);
        Draw(rp,220+modografico*70,140+z*3);

    /*  ...e una poligonale*/
        Move(rp,10,65);
        PolyDraw(rp,8,&drawvertici[0]);

    /*  Costruiamo un area quadrata e
        due circolari*/
        AreaMove(rp,150,20);
        AreaDraw(rp,210,20);
        AreaDraw(rp,210,80);
        AreaDraw(rp,150,80);
        AreaEllipse(rp,150,50,30,30);
        AreaEllipse(rp,150,50,15,15);
        AreaEnd(rp);

    /*  Costruiamo un rettangolo pieno*/
        RectFill(rp,223,20,300,80);
        if(Attesa()==2) goto fine;
        }/* z */
    }/* modografico*/

/*Riempiamo lo Screen col colore 15*/
SetRast(rp,15);

/*Disegnamo un cerchio...*/
SetDrMd(rp,1);
SetDrPt(rp,0xFFFF);
SetAPen(rp,10);
DrawEllipse(rp,150,100,90,90);

/*...e lo riempiamo con la maschera multicolore*/
SetAfPt(rp,&areamaskcol[0],-3);
SetAPen(rp,-1);
SetBPen(rp,0);
SetOPen(rp,10);
Flood(rp,0,150,100);
Attesa();

fine:
/* liberiamo le risorse allocate */
if(areabuffer) FreeMem(areabuffer,numareavt*5);
if(bitplane) FreeRaster(bitplane,largh,altez);
if(w)
  {
  while (im = (struct IntuiMessage *)
                          GetMsg(w->UserPort))
          ReplyMsg((struct Message *)im);
  CloseWindow(w);
  }
if(s) CloseScreen(s);
if(IntuitionBase) CloseLibrary(IntuitionBase);
if(GfxBase) CloseLibrary(GfxBase);
}

int Attesa()
{
UBYTE uscita=0;
while(uscita==0)
    {
        Wait(1 << w->UserPort->mp_SigBit);
        while(im = (struct IntuiMessage *)
                              GetMsg(w->UserPort))
        {
            if(im->Class==MOUSEBUTTONS &&
                    im->Code==SELECTDOWN) uscita=1;
            if(im->Class==CLOSEWINDOW) uscita=2;
            ReplyMsg((struct Message *)im);
        }
    }
return uscita;
}
```

Graphics.library 1.3

La lista che segue elenca le funzioni della libreria grafica ed alcune macro definite nei file include del linguaggio C. Accanto al nome di ogni funzione si riporta l'offset rispetto alla base della libreria in valori esadecimali e decimali. Si noti che la definizione dei parametri indica solo come debbano essere intese e utilizzate le variabili stesse e quale sia la loro parte significativa: esso NON indica la lunghezza effettiva dei parametri che è SEMPRE di 4 byte, anche quando si tratta di semplici char! Sullo stack, in C, viene posta sempre una longword e le funzioni ritornano sempre una longword. I prototipi delle funzioni per il Lattice C, ad esempio, definiscono solo due tipi di valori: puntatori e long; pertanto il compilatore convertirà ogni variabile diversamente definita, emettendo un warning in fase di compilazione. I caratteri posti al di sotto dei parametri indicano il registro in cui il parametro va passato alla funzione in linguaggio assembler; la cifra che segue, in qualche caso, il nome del registro (a0:16), indica quali bit del registro risultino significativi per la funzione chiamata.

AddAnimOb -- OFFSET: -$9c -156

Aggiunge un AnimOb alla lista indirizzata da "aop", inizializzando i campi Timer delle associate strutture AnimComp; per ogni bob compreso nell'AnimOb, chiama AddBob.

```
AddAnimOb( ao, aop, rp )
           a0  a1   a2
struct AnimOb *ao, **aop;
struct RastPort *rp;
```

AddBob -- OFFSET: -$60 -96

Aggiunge un bob alla lista dei gel.

```
AddBob( bob, rp )
        a0   a1
struct Bob *bob;
struct RastPort *rp;
```

AddFont -- OFFSET: -$1e0 -480

Aggiunge una font, residente in ram di tipo PUBLIC, alla lista di sistema.

```
AddFont( tf )
         a1
struct TextFont *tf;
```

AddVSprite -- OFFSET: -$66 -102

Aggiunge un VSprite alla lista dei gel, rispettando l'ordinamento determinato dai campi "VSprite.Y" e "VSprite.X".

```
AddVSprite( vs, rp )
            a0  a1
struct VSprite *vs;
struct RastPort *rp;
```

AllocRaster -- OFFSET: -$1ec -492

Alloca chip ram per un bitplane dalle dimensioni richieste (in pixel).

```
bitplane = AllocRaster( largh, altez )
   d0                   d0:16  d1:16
PLANEPTR bitplane;
USHORT largh, altez;
```

AndRectRegion -- OFFSET: -$1f8 -504

Effettua un AND logico fra il rettangolo e la regione, ponendo il risultato nella regione.

```
AndRectRegion( rg, rt )
               a0  a1
struct Region *rg;
struct Rectangle *rt;
```

AndRegionRegion -- OFFSET: -$270 -624

Effettua un AND logico fra le due regioni, ponendo il risultato nella seconda.

```
successo = AndRegionRegion( rg1, rg2 )
  d0:16                     a0   a1

BOOL successo;
struct Region *rg1, *rg2;
```

Animate -- OFFSET: -$a2 -162

Realizza l'animazione, aggiornando la lista degli AnimOb e dei loro componenti, chiamando le eventuali funzioni associate, modificando le posizioni di ogni VSprite.

```
Animate( aop, rp )
         a0   a1
struct AnimOb **aop;
struct RastPort *rp;
```

AreaCircle -- MACRO definita in "gfxmacros.h"

Aggiunge un area piena di forma circolare ad AreaInfo, occupandone due vertici: il centro è "cx", "cy", il raggio "r". Non usa la OPen; richiede AreaInfo e TmpRas.

```
errore = AreaCircle( rp, cx, cy, r )
LONG errore;
struct RastPort *rp;
SHORT cx, cy , r;

TESTO:
AreaEllipse( rp, cx, cy, r, r );
```

AreaDraw -- OFFSET: -$102 -258

Aggiunge un nuovo vertice ad AreaInfo per la costruzione di aree piene. Non usa LinePtrn; richiede AreaInfo e TmpRas.

```
errore = AreaDraw( rp, x,    y    )
  d0               a1  d0:16 d1:16
LONG errore;
struct RastPort *rp;
SHORT   x, y;
```

AreaEllipse -- OFFSET: -$ba -186

Aggiunge un'area piena di forma ellittica ad AreaInfo, occupandone due vertici, il centro è "cx", "cy", il raggio orizzontale "o", quello verticale "v"; non usa la OPen; richiede AreaInfo e TmpRas.

```
errore = AreaEllipse( rp, cx,   cy,   o,    v    )
  d0                  a1  d0:16 d1:16 d2:16 d3:16
LONG    errore;
struct  RastPort *rp;
SHORT   cx, cy, o, v;
```

AreaEnd -- OFFSET: -$108 -264

Aggiunge un vertice ad AreaInfo, ne elabora i valori per visualizzare aree piene, e riazzera il buffer; richiede AreaInfo e TmpRas.

```
errore = AreaEnd( rp )
  d0              a1
LONG errore;
struct RastPort *rp;
```

AreaMove -- OFFSET: -$fc -252

Aggiunge un vertice ad AreaInfo, defininendo l'inizio di una nuova area; chiude ogni eventuale area precedentemente iniziata, che starà in rapporto di XOR logico con quella nuova; richiede AreaInfo e TmpRas.

```
errore = AreaMove( rp, x,     y   )
  d0               a1  d0:16 d1:16
LONG errore;
struct RastPort *rp;
SHORT   x, y;
```

AskFont -- OFFSET: -$1da -474

Ritorna, nella struttura TextAttr, gli attributi della font corrente per la RastPort specificata.

```
AskFont( rp, ta )
         a1  a0
struct RastPort *rp;
struct TextAttr *ta;
```

AskSoftStyle -- OFFSET: -$54 -84

Ritorna una maschera che indica quali stili sono generabili, mediante algoritmo, nella font corrente.

```
maschera = AskSoftStyle( rp )
   d0                    a1
ULONG maschera;
struct RastPort *rp;
```

AttemptLockLayerRom -- OFFSET: -$28e -654

Ritorna un valore booleano che, in caso di successo, garantisce un accesso esclusivo al layer.

```
successo = AttemptLockLayerRom( l )
  d0:16                         a5
BOOL successo;
struct Layer *l;
```

BltBitMap -- OFFSET: -$1e -30

Muove una regione rettangolare di bit da una BitMap sorgente a una BitMap destinazione, applicando l'operazione logica definita in "minterm", sui piani di bit indicati in "mask"; se le due BitMap coincidono e i rettangoli si sovrappongono, "tempA" deve puntare a un buffer temporaneo di chip ram.

```
numpiani = BltBitMap( srgbm, srgx,  srgy, dstbm,
   d0                 a0    d0:16  d1:16   a1
dstx, dsty, largh, altez, minterm, mask [, tempA] )
d2:16 d3:16 d4:16  d5:16  d6:8     d7:8    [a2]

ULONG   numpiani;
struct  BitMap *srgbm;
SHORT   srgx, srgy;
struct  BitMap *dstbm;
SHORT dstx, dsty, largh, altez;
UBYTE   minterm, mask;
CPTR    tempA;
```

BltBitMapRastPort -- OFFSET: -$25e -606

Muove una regione rettangolare di bit da una BitMap sorgente a una RastPort destinazione, applicando l'operazione logica definita in "minterm".

```
BltBitMapRastPort( srgbm, srgx, srgy, dstrp, dstx,
                   a0     d0:16 d1:16  a1    d2:16
                  dsty, largh, altez, minterm )
                  d3:16 d4:16  d5:16  d6:8
struct  BitMap *srgbm;
SHORT   srgx, srgy;
struct  RastPort *dstrp;
SHORT   dstx, dsty, largh, altez;
UBYTE   minterm;
```

BltClear -- OFFSET: -$12c -300

Azzera il numero di byte della chip ram indicato dalla combinazione fra i parametri "flags" e "n", a partire dall'indirizzo "i", usando il blitter.

```
BltClear( i, n, flags )
          a1 d0  d1
APTR    i;
ULONG   n, flags;
```

BltMaskBitMapRastPort -- OFFSET: -$27c -636

Muove una regione rettangolare di bit da una BitMap sorgente a una RastPort destinazione, applicando l'operazione logica definita in "minterm", sui pixel corrispondenti ai bit diversi da 0 dell'area di chip ram indirizzata da "maschera".

```
BltMaskBitMapRastPort( srgbm, srgx, srgy, dstrp,
                       a0     d0:16 d1:16  a1
      dstx, dsty, largh, altez, minterm, maschera )
      d2:16 d3:16 d4:16  d5:16   d6:8      a2
struct  BitMap *srgbm;
SHORT   srgx, srgy;
struct  RastPort *dstrp;
SHORT   dstx, dsty, largh, altez;
UBYTE   minterm;
APTR    maschera;
```

BltPattern -- OFFSET: -$138 -312

Disegna col blitter, nel rettangolo specificato, la maschera puntata da "maschera" e larga un numero pari di byte ("n"), utilizzando i valori correnti delle penne, del modo grafico, di AreaPtrn e di "RastPort.Mask".

```
BltPattern( rp, maschera, dstx, dsty, dstxmax,
            a1, a0        d0:16 d1:16 d2:16
            dstymax, n )
             d3:16  d4:16
struct  RastPort *rp;
APTR    maschera;
SHORT   dstx, dsty, dstxmax, dstymax, n;
```

BltTemplate -- OFFSET: -$24 -36

Copia l'area rettangolare del bitplane che ha inizio al bit "offset" dell'indirizzo pari puntato da "srg", di larghezza, altezza e modulo specificati, alla locazione dstx, dsty, utilizzando il modo grafico e la RasPort.Mask correnti.

```
BltTemplate( srg, offset, modulo, rp, dstx, dsty,
             a0   d0:16   d1:16   a1  d2:16 d3:16
             largh, altez )
             d4:16  d5:16
CPTR    srg;
```

```
SHORT   offset, modulo;
struct  RastPort *rp;
SHORT   dstx, dsty, largh, altez;
```

BNDRYOFF -- MACRO definita in "gfxmacros.h"

Cancella il flag AREAOUTLINE nella RastPort, disabilitando l'orlatura delle aree piene.

```
BNDRYOFF( rp )
struct RastPort *rp;

TESTO:
{(rp)->Flags &= ~AREAOUTLINE;}
```

CBump -- OFFSET: -$16e -366

Prepara la UCopList a ricevere una nuova istruzione dell'utente.

```
CBump( c )
       a1
struct UCopList *c;
```

CEND -- MACRO definita in "gfxmacros.h"

Inserisce una istruzione di "end" nella UCopList, chiamando la macro CWAIT con i valori opportuni.

```
CEND( c )
struct UCopList *c;

TESTO:
{ CWAIT( c, 10000, 255 ); }
```

ChangeSprite -- OFFSET: -$1a4 -420

Usa i dati di chip ram puntati da "i" per costruire l'immagine dello sprite hardware.

```
ChangeSprite( vp, ss, i )
              a0  a1  a2
struct  ViewPort *vp;
struct  SimpleSprite *ss;
APTR    i;
```

CINIT -- MACRO definita in "gfxmacros.h"

Ritorna un puntatore a una UCopList inizializzata. Se "c" è nullo, alloca anche la memoria necessaria a contenere "n" istruzioni.

```
ucl = CINIT( c, n )
struct  UCopList *ucl, *c;
SHORT   n;

TESTO:
{ UCopperListInit( c, n ); }
```

ClearEOL -- OFFSET: -$2a -42

Pulisce una riga di pixel alta come la font corrente, a partire dalla posizione attuale e fino al margine destro della RastPort.

```
ClearEOL( rp )
          a1
struct RastPort *rp;
```

ClearRectRegion -- OFFSET: -$20a -522

Pulisce la porzione della regione che cade entro il rettangolo specificato, annullando così il risultato di ogni operazione precedente.

```
successo = ClearRectRegion( rg, rt )
  d0:16                     a0  a1
BOOL    successo;
struct  Region *rg;
struct  Rectangle *rt;
```

ClearRegion -- OFFSET: -$210 -528

Pulisce l'intera regione, annullando, così, il risultato di ogni operazione precedente.

```
ClearRegion( rg )
             a0
struct Region *rg;
```

ClearScreen -- OFFSET: -$30 -48

Pulisce la RastPort, ponendo a 0 tutti i bit, a partire dalla posizione corrente fino all'angolo in basso a destra.

```
ClearScreen( rp )
             a1
struct RastPort *rp;
```

ClipBlit -- OFFSET: -$228 -552

La funzione chiama BltBitMap() in maniera tale da tenere conto dei layer associati alle RastPorts; il suo uso appare preferibile nel contesto di Intuition.

```
ClipBlit( srgrp, srgx, srgy, dstrp, dstx, dsty,
          a0     d0:16 d1:16 a1     d2:16 d3:16
          largh, altez, minterm )
          d4:16  d5:16  d6:8
struct  RastPort *srgrp;
SHORT   srgx, srgy;
struct  RastPort *dstrp;
SHORT   dstx, dsty, largh, altez;
UBYTE   minterm;
```

CloseFont -- OFFSET: -$4e -78

Chiude una font aperta in precedenza con OpenFont, rilasciando le risorse associate.

```
CloseFont( tf )
           a1
struct TextFont *tf;
```

CMove -- OFFSET: -$174 -372

Aggiunge alla UCopList un istruzione del tipo: muovi il valore "v" nel registro hardware "a".

```
CMove( c, a, v )
       a1 d0 d1:16
struct  UCopList *c;
APTR    a;
SHORT   v;
```

CMOVE -- MACRO definita in "gfxmacros.h"

Aggiunge alla UCopList un istruzione del tipo CMove e prepara la lista per una nuova istruzione; il registro "r" deve essere espresso come campo della struttura custom.

```
CMOVE( c, r, v )
struct  UCopList *c;
APTR    r;
SHORT   v;

TESTO:
{ CMove( c, &r, v );CBump( c ); }
```

CopySBitMap -- OFFSET: -$1c2 -450

Copia tutti i bit della SuperBitMap entro i ClipRect del layer: è la

funzione opposta a SyncSBitMap.

```
CopySBitMap( l )
             a0
struct Layer *l;
```

CWait -- OFFSET: -$17a -378

Aggiunge alla UCopList un istruzione del tipo: attendi la posizione verticale ("v") e orizzontale ("h") del pennello elettronico.

```
CWait( c, v,     h )
       a1 d0:16 d1:16
struct UCopList *c;
SHORT v, h;
```

CWAIT -- MACRO definita in "gfxmacros.h"

Aggiunge alla UCopList un istruzione del tipo CWait e prepara la lista per la prossima istruzione.

```
CWAIT( c, v, h )
struct UCopList *c;
SHORT v, h;

TESTO:
{ CWait( c, v, h );CBump( c ); }
```

DisownBlitter -- OFFSET: -$1ce -462

Restituisce il blitter al sistema dopo una fase di uso esclusivo.

```
DisownBlitter()
```

DisposeRegion -- OFFSET: -$216 -534

Libera la memoria utilizzata dalla regione.

```
DisposeRegion( rg )
               a0
struct Region *rg;
```

DoCollision -- OFFSET: -$6c -108

Verifica se è avvenuta qualche collisione fra gel o fra gel e bordo e chiama le funzioni corrispondenti.

```
DoCollision( rp )
             a1
struct RastPort *rp;
```

Draw -- OFFSET: -$f6 -246

Disegna una linea dalla corrente posizione della penna al punto "x", "y", utilizzando il colore della penna corrente, il modo grafico e LinePtrn.

```
Draw( rp, x,     y )
      a1  d0:16 d1:16
struct RastPort *rp;
SHORT x, y;
```

DrawCircle -- MACRO definita in "gfxmacros.h"

Disegna un cerchio di centro "cx", "cy" e raggio "r", utilizzando la APen; non usa LinePtrn.

```
DrawCircle( rp, cx, cy, r )
struct RastPort *rp;
SHORT cx, cy, r;

TESTO:
DrawEllipse( rp, cx, cy, r, r );
```

DrawEllipse -- OFFSET: -$b4 -180

Disegna un ellisse di centro "cx", "cy", raggio orizzontale "o" e verticale "v", usando la APen; non usa LinePtrn.

```
DrawEllipse( rp, cx,     cy,     o,     v )
             a1  d0:16   d1:16   d2:16 d3:16
struct RastPort *rp;
SHORT cx, cy, o, v;
```

DrawGList -- OFFSET: -$72 -114

Elabora la lista dei gel istruendo il copper per la visualizzazione degli sprite e disegnando nella RastPort i bob.

```
DrawGList( rp, vp )
           a1  a0
struct RastPort *rp;
struct ViewPort *vp;
```

Flood -- OFFSET: -$14a -330

Riempe l'area che circonda il punto "x", "y" nel rispetto del corrente modo grafico e AreaPtrn; il parametro "modo" indica se il riempimento deve riguardare tutti i pixel adiacenti dello stesso colore di quello iniziale (1) o tutti quelli diversi dalla OPen (0). Richiede che TmpRas sia inizializzata.

```
errore = Flood( rp, modo, x,      y )
 d0:16          a1  d2    d0:16  d1:16
BOOL     errore;
struct  RastPort *rp;
ULONG   modo;
SHORT   x, y;
```

FreeColorMap -- OFFSET: -$240 -576

Libera la memoria allocata con la precedente chiamata a GetColorMap.

```
FreeColorMap( cm )
              a0
struct ColorMap *cm;
```

FreeCopList -- OFFSET: -$222 -546

Libera la memoria associata alla CopList.

```
FreeCopList( cl )
             a0
struct CopList *cl;
```

FreeCprList -- OFFSET: -$234 -564

Libera la memoria associata alla cprlist.

```
FreeCprList( crl )
             a0
struct cprlist *crl;
```

FreeGBuffers -- OFFSET: -$258 -600

Libera la memoria allocata con GetGBuffers, "db" indica se l'utente ha usato la tecnica del "double buffering".

```
FreeGBuffers( ao, rp, db )
              a0  a1  d0:16
struct  AnimOb *ao;
struct  RastPort *rp;
BOOL    db;
```

FreeRaster -- OFFSET: -$1f2 -498

Libera la memoria del bitplane allocata con AllocRaster.

```
FreeRaster( p, largh, altez )
            a0 d0:16  d1:16
```

```
PLANEPTR p;
USHORT   largh, altez;
```

FreeSprite -- OFFSET: -$19e -414

Rilascia lo sprite hardware "n", di cui si era assunto il controllo con una precedente chiamata a GetSprite, perché altri utenti possano accedervi.

```
FreeSprite( n )
            d0:16
SHORT n;
```

FreeVPortCopLists -- OFFSET: -$21c -540

Rilascia la memoria di tutte le liste copper di questa ViewPort.

```
FreeVPortCopLists( vp )
                   a0
struct ViewPort *vp;
```

GetColorMap -- OFFSET: -$23a -570

Alloca e inizializza una ColorMap e associata ColorTable capace di indirizzare "n" colori.

```
cm = GetColorMap( n )
d0                d0
struct ColorMap *cm;
LONG n;
```

GetGBuffers -- OFFSET: -$a8 -168

Alloca la memoria per tutti i buffer (SaveBuffer, BorderLine, CollMask, ImageShadow e, se "db" lo richiede, anche DBufPacket e BufBuffer) per ogni componente dell'AnimOb.

```
successo = GetGBuffers( ao,  rp, db )
  d0:16                 a0   a1  d0:16
BOOL successo;
struct AnimOb *ao;
struct RastPort *rp;
BOOL db;
```

GetRGB4 -- OFFSET: -$246 -582

Ritorna il valore RGB del colore "c": il blu nei bit 0-3, il verde nei bit 4-7, il rosso nei bit 8-11; il formato potrà cambiare in futuro.

```
rgb = GetRGB4( cm, c )
d0             a0  d0
ULONG rgb;
struct ColorMap *cm;
LONG c;
```

GetSprite -- OFFSET: -$198 -408

Ritorna un valore che indica se la richiesta "r" di usare uno sprite, è stata soddisfatta dal sistema.

```
n = GetSprite( ss, r )
d0:16          a0  d0:16
SHORT   n;
struct  SimpleSprite *ss;
SHORT   r;
```

InitAnimate -- MACRO definita in "gels.h"

Inizializza il sistema di animazione.

```
InitAnimate( aop )
struct  AnimOb **aop;

TESTO:
{*(aop) = NULL;}
```

InitArea -- OFFSET: -$11a -282

Inizializza la struttura AreaInfo, in modo che possa contenere "n" vertici: il buffer di chip ram, che deve iniziare ad indirizzo pari, deve essere lungo almeno 5 * "n" byte.

```
InitArea( ai, buffer, n )
          a0  a1      d0:16
struct AreaInfo *ai;
APTR buffer;
SHORT n;
```

InitBitMap -- OFFSET: -$186 -390

Inizializza la struttura BitMap in modo che possa gestire "prof" bitplane della larghezza e altezza richiesta.

```
InitBitMap( bm, prof, largh, altez )
            a0  d0:8  d1:16  d2:16
struct  BitMap *bm;
BYTE    prof;
SHORT   largh, altez;
```

InitGels -- OFFSET: -$78 -120

Inizializza la struttura GelsInfo con due VSprite che fungono da testa e da coda della lista dei gel.

```
InitGels( testa, coda, gi )
          a0     a1    a2
struct VSprite *testa, *coda;
struct GelsInfo *gi;
```

InitGMasks -- OFFSET: -$ae -174

Inizializza la struttura AnimOb, chiamando per ogni componente la funzione InitMask.

```
InitGMasks( ao )
            a0
struct AnimOb *ao;
```

InitMasks -- OFFSET: -$7e -126

Inizializza le maschere BorderLine e CollMask di una struttura VSprite.

```
InitMasks( vs )
           a0
struct VSprite *vs;
```

InitRastPort -- OFFSET: -$c6 -198

Inizializza la struttura RastPort.

```
InitRastPort( rp )
              a1
struct RastPort *rp;
```

InitTmpRas -- OFFSET: -$1d4 -468

Inizializza la struttura TmpRas, utilizzata dalle funzioni per la costruzione e il riempimento di aree; il bitplane puntato da "bitplane", deve essere abbastanza ampio da accogliere le immagini create dalle funzioni grafiche: la sua lunghezza in byte viene calcolata dalla macro RASSIZE.

```
tmpras = InitTmpRas( tr, bitplane, lungh )
  d0                 a0  a1         d0
struct TmpRas *tmpras, *tr;
PLANEPTR bitplane;
LONG lungh;
```

InitView -- OFFSET: -$168 -360

Inizializza la struttura View.

```
InitView( v )
          a1
struct View *v;
```

InitVPort -- OFFSET: -$cc -204

Inizializza la struttura ViewPort.

```
InitVPort( vp )
           a0
struct ViewPort *vp;
```

LoadRGB4 -- OFFSET: -$c0 -192

Carica "n" valori RGB, contenuti nell'array indirizzato da "rgb", a partire dal colore 0 della tavola dei colori associata alla ViewPort.

```
LoadRGB4( vp, rgb, n )
          a0  a1   d0:16
struct  ViewPort *vp;
UWORD   *rgb;
SHORT   n;
```

LoadView -- OFFSET: -$de -222

Utilizza la lista di istruzioni per il copper, contenuta nella View, per costruire il display corrente.

```
LoadView( v )
          a1
struct View *v;
```

LockLayerRom -- OFFSET: -$1b0 -432

Inibisce a tutti gli altri task l'accesso alla struttura Layer.

```
LockLayerRom( l )
              a5
struct Layer *l;
```

MakeVPort -- OFFSET: -$d8 -216

Trasforma le informazioni contenute nelle strutture View, ViewPort e RasInfo in una lista provvisoria di istruzioni per il copper.

```
MakeVPort( v, vp )
           a0 a1
struct View *v;
struct ViewPort *vp;
```

Move -- OFFSET: -$f0 -240

Stabilisce in "x", "y", la nuova posizione della penna grafica, relativa all'angolo in alto a sinistra della RastPort.

```
Move( rp, x,     y )
      a1  d0:16 d1:16
struct RastPort *rp;
SHORT x, y;
```

MoveSprite -- OFFSET: -$1aa -426

Muove lo sprite hardware nella posizione "x", "y", relativa all'angolo in alto a sinistra della ViewPort (un bug della routine fa sì che lo sprite appaia un pixel a sinistra di quanto specificato).

```
MoveSprite( vp, ss, x, y )
            a0  a1  d0 d1
struct  ViewPort *vp;
struct  SimpleSprite *ss;
SHORT   x, y;
```

MrgCop -- OFFSET: -$d2 -210

Unifica tutte le liste provvisorie di istruzioni per il copper in un'unica lista pronta per essere utilizzata.

```
MrgCop( v )
        a1
struct View *v;
```

NewRegion -- OFFSET: -$204 -516

Alloca e inizializza una nuova struttura Region.

```
rg = NewRegion()
d0
struct Region *rg;
```

OpenFont -- OFFSET: -$48 -72

Se possibile, rende disponibile, per l'uso con le routine grafiche, la font descritta nella struttura TextAttr.

```
tf = OpenFont( ta )
d0             a0
struct TextFont *tf;
struct TextAttr *ta;
```

OrRectRegion -- OFFSET: -$1fe -510

Aggiunge il rettangolo alla regione attraverso un'operazione di OR logico.

```
successo = OrRectRegion( rg, rt )
 d0:16                   a0  a1
BOOL successo
struct Region *rg;
struct Rectangle *rt;
```

OrRegionRegion -- OFFSET: -$264 -612

Aggiunge alla seconda regione i valori della prima attraverso un OR logico.

```
successo = OrRegionRegion( rg1, rg2 )
 d0:16                     a0   a1
BOOL    successo;
struct  Region *rg1, *rg2;
```

OwnBlitter -- OFFSET: -$1c8 -456

Ritorna il controllo al chiamante quando il blitter è in procinto di diventare disponibile per un uso esclusivo; la funzione WaitBlit assicura, poi, che il blitter possa essere effetivamente usato.

```
OwnBlitter()
```

PolyDraw -- OFFSET: -$150 -336

Collega, con una linea spezzata continua, gli "n" vertici le cui coordinate si trovano nell'array indirizzato da "vrt".

```
PolyDraw( rp, n,    vrt )
          a1  d0:16 a0
struct  RastPort *rp;
SHORT   n, *vrt;
```

QBlit -- OFFSET: -$114 -276

Aggiunge una richiesta di uso del blitter alla lista di sistema; la funzione indirizzata dalla struttura bltnode verrà chiamata quando tutte le richieste precedenti saranno state soddisfatte.

```
QBlit( bn )
       a1
struct bltnode *bn;
```

QBSBlit -- OFFSET: -$126 -294

Ritorna al chiamante quando il pennello elettronico ha raggiunto una data posizione e il blitter è disponibile per l'utente.

```
QBSBlit( bn )
         a1
struct  bltnode *bn;
```

RASSIZE -- MACRO definita in "gfx.h"

Calcola la lunghezza in byte di un bitplane; deve essere un multiplo di 2.

```
lunghezza = RASSIZE( largh, altez )
LONG lunghezza;
USHORT largh, altez;

TESTO:
(( altez ) * (( largh+15 ) > 3 & 0xFFFE))
```

ReadPixel -- OFFSET: -$13e -318

Ritorna il colore "c" del pixel che si trova nella locazione "x", "y" relativa all'angolo in alto a sinistra della RastPort.

```
c = ReadPixel( rp, x,     y )
d0          a1  d0:16 d1:16
LONG    c;
struct  RastPort *rp;
SHORT   x, y;
```

RectFill -- OFFSET: -$132 -306

Riempe il rettangolo definito rispettando i parametri impostati nella RastPort.

```
RectFill( rp, x,     y,     xmax, ymax )
          a1  d0:16 d1:16 d2:16 d3:16
struct  RastPort *rp;
SHORT   x, y, xmax, ymax;
```

RemBob -- MACRO definita in "gels.h"

Avverte il sistema di rimuovere il bob dalla lista dei gel. Il bob sarà effettivamente rimosso alla prossima chiamata a DrawGList.

```
RemBob( bob )
struct Bob *bob;

TESTO:
{(bob)->Flags |= BOBSAWAY;}
```

RemFont -- OFFSET: -$1e6 -486

Rimuove la font dalla lista di sistema. I successivi tentativi di utilizzarla chiamando SetFont non verranno soddisfatti.

```
RemFont( tf )
         a1
struct TextFont *tf;
```

RemIBob -- OFFSET: -$84 -132

Rimuove immediatamente il bob dalla lista dei gel e lo cancella dalla RastPort.

```
RemIBob( bob, rp, vp )
         a0   a1  a2
struct Bob *bob;
struct RastPort *rp;
struct ViewPort *vp;
```

RemVSprite -- OFFSET: -$8a -138

Rimuove il VSprite dalla lista dei gel.

```
RemVSprite( vs )
            a0
struct VSprite *vs;
```

ScrollRaster -- OFFSET: -$18c -396

Muove dei fattori "dx", "dy" verso la locazione 0, 0, i bit del raster contenuti nel rettangolo specificato. Lo spazio rimasto vuoto viene riempito con la BPen.

```
ScrollRaster( rp, dx,    dy,    x,    y,    xmax,
              a1  d0:16  d1:16  d2:16 d3:16 d4:16
              ymax )
              d5:16
struct RastPort *rp;
SHORT dx, dy, x, y, xmax, ymax;
```

ScrollVPort -- OFFSET: -$24c -588

Ricostruisce le liste del copper relative alla ViewPort per riflettere i valori "RasInfo.RxOffset" e "RasInfo.RyOffset".

```
ScrollVPort( vp )
             a0
struct ViewPort *vp;
```

SetAfPt -- MACRO definita in "gfxmacros.h"

Definisce la maschera da utilizzare per le aree piene; la maschera deve consistere in un rettangolo largo 16 bit (USHORT) e alto 2 elevato "e" pixel; se "e" è negativo la maschera è multicolore.

```
SetAfPt( rp, maschera, e )
struct RastPort *rp;
USHORT *maschera;
BYTE e;

TESTO:
{(rp)->AreaPtrn = maschera;(rp)->AreaPtSz = e;}
```

SetAPen -- OFFSET: -$156 -342

Associa alla APen (o FgPen) il colore "c" della ColorTable.

```
SetAPen( rp, c )
         a1  d0:8
struct RastPort *rp;
UBYTE c;
```

SetBPen -- OFFSET: -$15c -348

Associa alla BPen (o BgPen) il colore "c" della ColorTable.

```
SetBPen( rp, c )
         a1  d0:8
struct  RastPort *rp;
UBYTE   c;
```

SetCollision -- OFFSET: -$90 -144

Associa alla posizione "i" dell'array dei puntatori alle routine di gestione delle collisioni, la funzione dell'utente.

```
SetCollision( i, funzione, gi )
              d0 a0        a1
ULONG   i;
VOID    (*funzione)();
struct  GelsInfo *gi;
```

SetDrMd -- OFFSET: -$162 -354

Stabilisce un nuovo modo grafico.

```
SetDrMd( rp, modo )
         a1  d0:8
struct RastPort *rp;
UBYTE modo;
```

SetDrPt -- MACRO definita in "gfxmacros.h"

Definisce la maschera per il disegno delle linee.

```
SetDrPt( rp, maschera )
struct RastPort *rp;
UWORD maschera;

TESTO:
{(rp)->LinePtrn = maschera;
 (rp)->Flags |= FRST_DOT; (rp)->linpatcnt=15;}
```

SetFont -- OFFSET: -$42 -66

Stabilisce la font corrente e i relativi attributi.

```
SetFont( rp, tf )
         a1  a0
struct RastPort *rp;
struct TextFont *tf;
```

SetOPen -- MACRO definita in "gfxmacros.h"

Associa alla OPen (o AOlPen) il colore "c" della ColorTable, e setta il flag (AREAOUTLINE) che obbliga il sistema a circondare le aree piene con un bordo colorato.

```
SetOPen( rp, c )
struct RastPort *rp;
UBYTE c;

TESTO:
{(rp)->AOlPen = c;(rp)->Flags |= AREAOUTLINE;}
```

SetRast -- OFFSET: -$ea -234

Riempe l'intera area grafica della RastPort con il colore "c".

```
SetRast( rp, c )
         a1  d0:8
struct  RastPort *rp;
UBYTE   c;
```

SetRGB4 -- OFFSET: -$120 -288

Carica il colore "c" della ColorMap associata alla ViewPort con i valori RGB specificati.

```
SetRGB4( vp, c,    rosso, verde, blu )
         a0  d0:16 d1:4   d2:4   d3:4
struct  ViewPort *vp;
SHORT   c;
UBYTE   rosso, verde, blu;
```

SetRGB4CM -- OFFSET: -$276 -630

Carica il colore "c" della ColorMap con i valori RGB specificati.

```
SetRGB4CM( cm, c,    rosso, verde, blu )
           a0  d0:16 d1:4   d2:4   d3:4
struct  ColorMap *cm;
SHORT   c;
UBYTE   rosso, verde, blu;
```

SetSoftStyle -- OFFSET: -$5a -90

Stabilisce lo stile della font corrente secondo le richieste dell'utente ("r") e le possibilità offerte dalla font (espresse da "maschera" che è il valore ritornato da AskSoftStyle).

```
stile = SetSoftStyle( rp, r, maschera )
 d0                   a1  d0 d1
ULONG   stile;
struct  RastPort *rp;
ULONG   r, maschera;
```

SetWrMsk -- MACRO definita in "gfxmacros.h"

Definisce quali piani di bit (posti ad 1 nel parametro "maschera") possono essere modificati dalle funzioni grafiche.

```
SetWrMsk( rp, maschera )
struct RastPort *rp;
UBYTE maschera;

TESTO:
{(rp)->Mask = maschera;}
```

SortGList -- OFFSET: -$96 -150

Ordina la lista dei gel secondo il valore delle coordinate di ogni elemento.

```
SortGList( rp )
           a1
struct RastPort *rp;
```

SyncSBitMap -- OFFSET: -$1bc -444

Copia tutti i bit dei ClipRects del layer entro la SuperBitMap: è la funzione opposta a CopySBitMap.

```
SyncSBitMap( l )
             a0
struct Layer *l;
```

Text -- OFFSET: -$3c -60

Scrive la stringa "str" di "n" caratteri a partire dalla posizione corrente della RastPort, utilizzando il modo grafico, la font e gli attributi ivi specificati.

```
Text( rp, str, n )
      a1  a0   d0:16
struct  RastPort *rp;
STRPTR  str;
SHORT   n;
```

TextLength -- OFFSET: -$36 -54

Ritorna la lunghezza misurata in pixel della stringa "str" di "n" caratteri, quando essa venisse scritta con la font e gli attributi correnti.

```
pixel = TextLength( rp, str, n )
d0:16               a1  a0   d0:16
SHORT pixel;
struct RastPort *rp;
STRPTR str;
SHORT n;
```

UCopperListInit -- OFFSET: -$252 -594

Ritorna un puntatore a una UCopList inizializzata; se "c" è nullo, alloca anche la memoria necessaria a contenere "n" istruzioni.

```
ucl = UCopperListInit( c, n )
d0                     a0 d0:16
struct  UCopList *ucl, *c;
SHORT   n;
```

UnlockLayerRom -- OFFSET: -$1b6 -438

Rende il layer nuovamente disponibile ad altri eventuali task.

```
UnlockLayerRom( l )
                a5
struct Layer *l;
```

VBeamPos -- OFFSET: -$180 -384

Ritorna la posizione verticale del pennello elettronico: essa risulta sufficientemente esatta solo se il task chiamante ha la più alta priorità.

```
pos = VBeamPos()
d0
LONG pos;
```

WaitBlit -- OFFSET: -$e4 -228

Ritorna quando il blitter ha concluso l'operazione in corso.

```
WaitBlit()
```

WaitBOVP -- OFFSET: -$192 -402

Ritorna quando il pennello elettronico ha raggiunto il fondo della ViewPort; il numero di task concorrenti incide sul momento dell'effettivo ritorno; un bug, inoltre, fa sì che questa funzione rallenti fortemente l'intero sistema.

```
WaitBOVP( vp )
          a0
struct ViewPort *vp
```

WaitTOF -- OFFSET: -$10e -270

Ritorna quando il pennello elettronico ha raggiunto il "vertical blank" e tutte le funzioni associate a questo interrupt hanno concluso il loro lavoro.

```
WaitTOF()
```

WritePixel -- OFFSET: -$144 -324

Usa la APen per modificare il colore del pixel "x", "y"; se esso si trova al di fuori dei confini della RastPort, ritorna -1.

```
errore = WritePixel(  rp, x,      y )
  d0                  a1  d0:16   d1:16
LONG errore;
struct RastPort *rp;
SHORT x, y;
```

XorRectRegion -- OFFSET: -$22e -558

Aggiunge il rettangolo alla regione attraverso un'operazione di XOR logico.

```
successo = XorRectRegion( rg, rt )
 d0:16                    a0  a1
BOOL successo;
struct Region *rg;
struct Rectangle *rt;
```

XorRegionRegion -- OFFSET: -$26a -618

Aggiunge alla seconda regione i valori della prima attraverso un XOR logico.

```
successo = XorRegionRegion( rg1, rg2 )
 d0:16                      a0   a1
BOOL successo;
struct Region *rg1, *rg2;
```

(segue da pagina 26)

```
/************************************************/

   char *
tabstr(tab)

{
static char space[80];
static int lasttab;
static int init = 1;
int i;

   if (init)

      {
      for (i=0; i < 80; i++)
         space[i] = ' ';
      space[init = 0] = '\0';
      }

   if (tab != lasttab)

      {
      space[lasttab] = ' ';
      space[lasttab = tab] = '\0';
      }

   return (space);

} /* end of tabstr */

/************************************************/

   void
chkread(ptr, size)
   UBYTE *ptr;
   int size;

{

   if (fread(ptr, size, 1, file) != 1)

      {
      printf("\nErrore nella lettura del file.\n");
      exit(20);
      }

} /* end of chkread */

/************************************************/
```

GRANDE CONCORSO 9 LIBRI JACKSON PER IL TUO AMIGA

PER TE FAVOLOSI PREMI

AMIGA DOS (con disco)
Cod. CC815 L. 59.000
AMIGA HANDBOOK
Cod. CC320 L. 35.000
IL MANUALE DI AMIGA
Cod. CZ532 L. 39.000
AMIGA 500 Guida all'acquisto
Cod. CC627 L. 55.000
AMIGA ASSEMBLER (con disco)
Cod. CL757 L. 59.000
AMIGA LINGUAGGIO C (con disco)
Cod. CL758 L. 52.000
AMIGA GRAFICA 3-D (con disco)
Cod. CZ756 L. 59.000
AMIGA BASIC (con disco)
Cod. CL768 L. 57.000

AMIGA - Tecniche di programmazione (con disco)
Cod. CC795 L. 62.000

Ecco una nuova straordinaria iniziativa del Gruppo Editoriale Jackson: è lo speciale concorso riservato a te e dedicato al tuo fantastico Amiga.

Partecipare è facile: basta acquistare uno dei nove libri Jackson per i computer Amiga, compilare la speciale cartolina che trovi dal tuo rivenditore di fiducia e spedirla.

E' facile anche vincere e i premi in palio sono fantastici: un personal computer Commodore Amiga 2000, una stampante a colori Commodore MPS 1500C e 8 set di programmi software della serie "Software Commodore by CTO".

GRUPPO EDITORIALE JACKSON

AUT. MIN. RICH.

IL PROGRAMMA PIU' EVOLUTO PER LO SVILUPPO DI PRONOSTICI A CONCORSO

TOTO

Amiga

3 PROGRAMMI IN UNO

E' IN EDICOLA

PER AMIGA 500 • 1000 • 2000

MEMORIZZAZIONE SETTIMANALE • SISTEMI A TUTTO CAMPO • SISTEMI A SEZIONI • SISTEMI INTEGRALI O SEMI-INTEGRALI • FORMULE DERIVATE • CONSECUTIVITA' DEI SEGNI • COLONNE CONDIZIONATE • INTERRUZIONI GENERALI E PARTICOLARI • SEQUENZE DI SEGNI A PASSO VARIABILE E CON MINIMO E MASSIMO CONSECUTIVO • ANALISI DEL SISTEMA IN BASE A FORMULE DERIVATE E QUANTITA' DI SEGNI • MEMORIZZAZIONE DEI SISTEMI SU DISCO • SPOGLIO ELETTRONICO • STAMPA VIDEO •

GRUPPO EDITORIALE JACKSON