Programmare in C su Amiga (28)

di Dario de Judicibus (MC2120)

Prima puntata dedicata ai controlli, cioè ad i famigerati gadget. Vedremo i vari tipi di controlli che Intuition ci mette a disposizione e la struttura base comune a tutti. Nella Scheda Tecnica altri cinque comandi dell'AmigaDOS 1.3

Introduzione

I controlli [gadget] sono il meccanismo principale di interazione tra l'utente ed Intuition, insieme ad i menu, già visti alcune puntate fa. Al contrario di quest'ultimi, che permettono all'utente di scegliere uno o più elementi in una lista strutturata gerarchicamente su tre livelli (menu, voci e sottovoci), i controlli rappresentano, in un'ottica di interfaccia orientata agli oggetti [*object-oriented interface*], quei meccanismi di controllo che usiamo nella vita di tutti i giorni per operare con le varie macchine di cui ci siamo circondati. Accendere una luce, spingere il pedale di un acceleratore, battere il codice di accesso su una tastiera Bancomat, sono oramai operazioni naturali, proprio perché oggigiorno siamo abituati ad una serie di controlli standard per forma e funzionamento. E la forma è fondamentale, perché è quella che fornisce all'utente l'informazione su come operare con quel determinato controllo. Due esempi fra i tanti. Un anno fa mi trovavo negli Stati Uniti. Era mezzanotte ed ero finalmente atterrato a San Francisco e, dopo aver superato la dogana, aver ritirato i bagagli, ed aver trovato la navetta giusta per il parcheggio della Hertz, mi trovavo in uno di quei macchinoni che piacciono tanto agli americani con l'intenzione di arrivare in albergo al più presto possibile. Ebbene, mi ci vollero almeno dieci minuti prima di partire, e non certo per il cambio automatico a cui ero già abituato. Il motivo erano i fari. Già, perché in quel particolare modello, i fari si accendevano non con una leva od un pulsante collocato sul cruscotto comandi o comunque vicino al volante, ma tirando un pomello malignamente nascosto in basso a sinistra, nella posizione cioè dove in molte macchine europee è posizionata la leva per l'apertura del cofano motore. Come se non bastasse, lo stesso pomello controllava l'intensità dell'illuminazione del cruscotto con i vari indicatori (tachimetro, carburante, ecc.). Come? Ma ruotandolo, ovviamente! Sarà stato complice anche il viaggio di sedici ore diretto da Roma, ma indubbiamente la non familiarità alla relazione forma/uso dei comandi è stato uno degli elementi determinanti nel rendere difficile ciò che probabilmente per un americano sarebbe stato probabilmente banale.

Un altro esempio lo possiamo trovare anche qui in Italia. Avete mai visto quelle lampade da tavolo a braccio pieghevole che si accendono ruotando un pirolino in cima al cono paralume? Beh, io ne ho una a casa, e vi posso assicurare che sette persone su dieci cercano di accenderla *premendolo*, poi *tirandolo*, ma solo dopo molti sforzi il 50% arriva a capire che bisogna girarlo. E naturalmente cercano poi di spegnerla girando nel verso opposto... non nello stesso verso, come invece l'ideatore del marchingegno diabolico ha previsto.

Cosa c'entra tutto ciò con la programmazione dei controlli? Semplice. Quello che intendo evidenziare è l'importanza nelle interfacce OO (cioè object-oriented, come le chiameremo d'ora in poi) di curare il rapporto tra la forma del controllo ed il suo utilizzo, e di cercare di mantenere questo rapporto coerentemente per tutta l'interfaccia, evitando *licenze poetiche* e *variazioni artistiche* che finiscono per confondere chi usa il vostro programma. Questa raccomandazione è molto importante nella versione 1.3 del sistema in quanto l'Amiga, al contrario per quanto avviene per il Mac, non fornisce al programmatore un set di strumenti standard di alto livello per costruire le proprie interfacce interattive [*ToolBox*], ma oggetti elementari fortemente personalizzabili. Se questo da una parte vi permette di sbizzarrirvi a creare le interfacce più complesse ed originali possibili, dall'altra porta ad una proliferazione di interfacce che confonde l'utente finale e dà una sensazione di poca professionalità che si ripercuote su tutto il sistema. Per questo motivo gli sviluppatori di sistemi dotati di interfacce OO ci tengono moltissimo che i programmatori si attengano a regole ben precise nel disegnare le interfacce dei loro prodotti (le cose sono in parte destinate a cambiare con l'avvento della nuova versione 2.0, comunque).

Regola numero uno, quindi: *se state disegnando un controllo che esiste già fra quelli di sistema, cercate di disegnarlo allo stesso modo, seguendo cioè lo stesso schema.*

Un esempio degli effetti negativi della mancanza di standardizzazione nell'Amiga è il *file requester*. Non esistendo come oggetto di sistema, ogni programma si è fatto il suo. Ora, non è tanto

```
struct Gadget
{
  struct Gadget    *NextGadget    ; /* Controllo successivo nella lista     */
  SHORT             LeftEdge      ; /* [ Dimensioni e posizione dell'area ] */
  SHORT             TopEdge       ; /* [ di selezione del controllo, in   ] */
  SHORT             Width         ; /* [ valori assoluti o relativi come  ] */
  SHORT             Height        ; /* [ specificato nel campo Flags      ] */
  USHORT            Flags         ; /* Attributi del controllo - vedi testo */
  USHORT            Activation    ; /* Modi di attivazione e selezione      */
  USHORT            GadgetType    ; /* Tipo di controllo                    */
  APTR              GadgetRender  ; /* Puntatore alla grafica del controllo */
  APTR              SelectRender  ; /* Come sopra, ma in caso di selezione  */
  struct IntuiText *GadgetText    ; /* Eventuale testo associato            */
  LONG              MutualExclude ; /* < Riservato per usi futuri           */
  APTR              SpecialInfo   ; /* Estensione della struttura (tipi)    */
  USHORT            GadgetID      ; /* Per identificare il controllo        */
  APTR              UserData      ; /* Eventuali dati utente                */
};
```

Figura 1
La struttura Gadget.

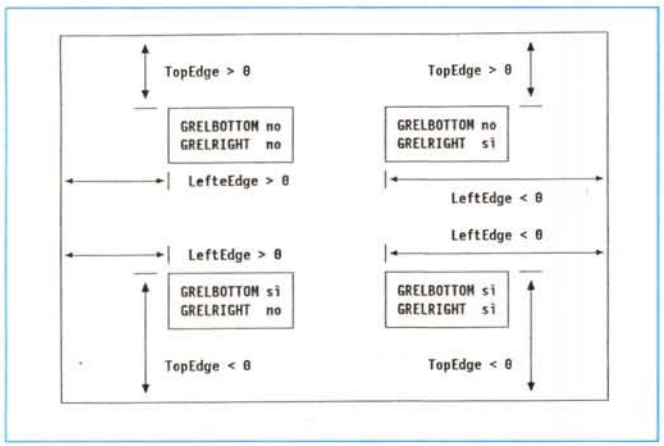

Figura 2 - Posizionamento di un controllo.

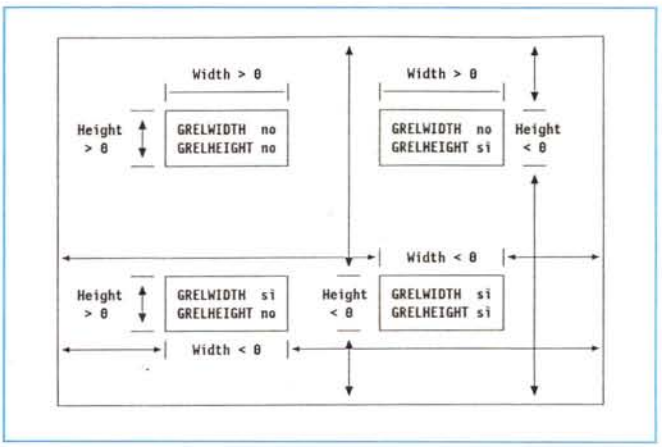

Figura 3 - Dimensioni di un controllo.

importante che i vari quadri siano differenti per forma e disposizione dei singoli elementi che li compongono, quanto il fatto che anche il *comportamento*, cioè il modo di reagire alle azioni dell'utente, spesso varia. Ed è questo che maggiormente confonde l'utente.

Ne viene la regola numero due: *assicuratevi che la vostra interfaccia sia sempre prevedibile dall'utente finale, che questi cioè sia sempre in grado di sapere cosa aspettarsi dal programma a fronte di una qualunque operazione.*

Non lasciatevi tentare dal desiderio di essere originali a tutti i costi. I programmatori dilettanti tendono spesso all'anarchia ed alla personalizzazione dei propri programmi, ma a meno che ciò non si materializzi in un effettivo vantaggio per l'utente finale, se cioè non ci sono ragioni che vanno al di là del semplice fatto estetico, si tratta sempre di un fattore negativo che si ripercuote sul favore che il mercato darà al vostro prodotto.

I controlli

Un controllo non è altro che un'area selezionabile [*select box*] contenuta in un oggetto grafico [*display element*] detto appunto *contenitore*. Esistono due tipi di controlli: quelli di *sistema*, di cui abbiamo già parlato molte puntate fa, in relazione agli schermi ed alle finestre, e quelli *delle applicazioni*, cioè quelli che ogni programma definisce per i propri scopi. Al momento, tutti i controlli di sistema sono sempre situati nei bordi di schermi o finestre, mentre quelli delle applicazioni non hanno tale limitazione, ma non possono essere definiti per gli schermi. Se si vuole simulare un controllo non di sistema in uno schermo, è necessario utilizzare una finestra di sfondo [*backdrop window*] di cui abbiamo già parlato.

L'utente attiva il controllo posizionando il puntatore del mouse nell'area di selezione e premendo il tasto sinistro. A questo punto l'interazione può proseguire in modo differente a seconda del tipo di controllo. Ad esempio, il controllo può essere formato da un cursore che scorre verticalmente lungo un binario. Tale cursore può essere agganciato con il mouse e, sempre tenendo premuto il tasto sinistro del mouse, può essere spostato a piacimento e quindi posizionato rilasciando il tasto di selezione. Tutte queste operazioni si riflettono nel programma in una serie di dati che possono essere analizzati e convertiti in informazioni che il programma può quindi utilizzare rispondendo così alla richiesta dell'utente, ad esempio facendo scorrere su e giù il testo di un file che è visualizzato nella finestra che contiene il controllo in oggetto.

Una cosa è molto importante da capire. L'area di selezione, che è quella poi che veramente conta ai fini dell'interazione con l'utente, non ha necessariamente alcuna relazione con l'aspetto esteriore del controllo, se non quella definita dal programmatore. In pratica è possibile «vestire» un controllo con un bordo od immagine di qualunque forma e dimensione, arrivando persino a disgiungere completamente l'area di selezione e l'immagine del controllo. Ovviamente una cosa del genere ha poco senso e serve solo a confondere l'utente. È tuttavia importante rendersi conto che il controllo *non* è la sua immagine, bensì quella parte del contenitore che può essere selezionata ed alla quale il bordo o l'immagine serve solo a dare visibilità.

Esistono tre differenti tipi di controlli, tutti molto elementari, con i quali cioè si possono costruire controlli composti praticamente di ogni tipo, e tutti che si basano su una singola struttura riportata in figura 1.

Pulsante

un pulsante [*Boolean gadget*] è un controllo di tipo *vero o falso*, cioè esso può riflettere solo uno di questi due stati;

controllo proporzionale

un controllo proporzionale [*Proportional gadget*] permette di impostare e/o visualizzare un valore all'interno di un range definito, in modo *analogico*;

campo di immissione/emissione dati

un campo [*String gadget*] permette di impostare e/o visualizzare una stringa di caratteri od un intero.

Nelle prossime puntate avremo modo di analizzare in dettaglio ognuno di questi tipi di controlli, sia per quello che riguarda la loro definizione in un programma, sia per quello che riguarda i vari modi di utilizzarli, le potenzialità ed i limiti di ogni singolo tipo nella attuale versione di Intuition. Non tratteremo per il momento la versione 2.0, anche perché ancora non così diffusa come la precedente.

La struttura Gadget

Questa struttura serve a definire qualunque tipo di controllo, salvo utilizzare una apposita estensione per alcuni tipi che necessitano di ulteriori informazioni per essere definiti. Essa ha ben quindici campi.

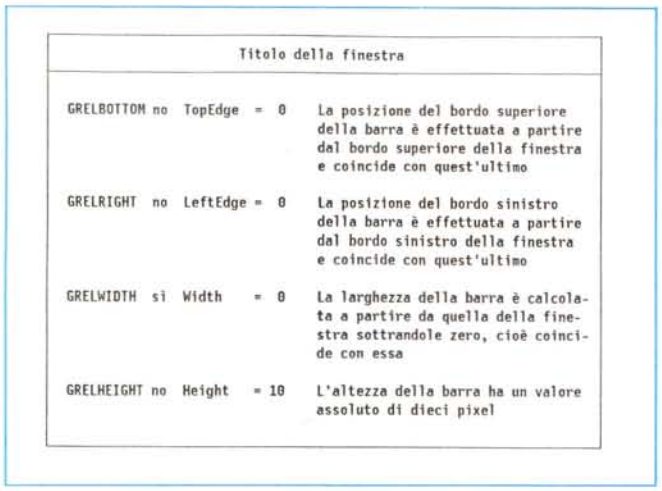

Figura 4 - Caratteristiche di una barra di spostamento a tutta ampiezza.

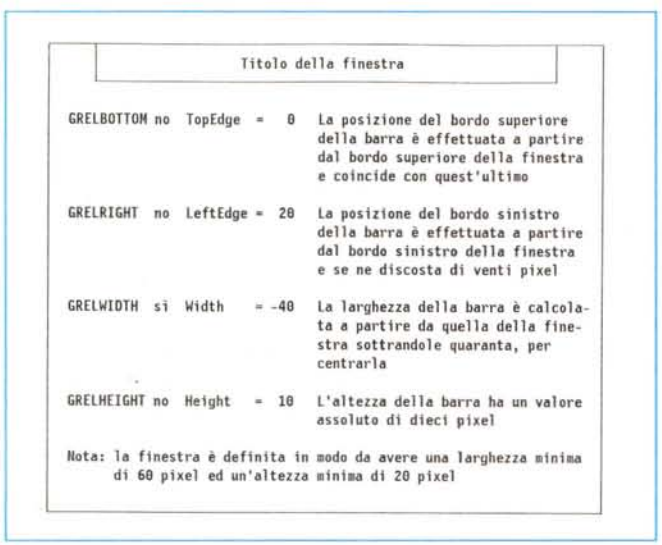

Figura 5
Caratteristiche di una barra di spostamento ad ampiezza ridotta.

```
/* ********************************************************************** **
** Attributi definibili da programma                                      **
** ********************************************************************** */
/*
** Tecnica di evidenziazione
*/
#define GADGHIGHBITS  0x0003  /* Bit utilizzati per l'evidenziazione       */
#define GADGHCOMP     0x0000  /* Usa il colore complementare per l'area    */
#define GADGHBOX      0x0001  /* Disegna un rettangolo intorno all'immagine */
#define GADGHIMAGE    0x0002  /* Scambia con l'immagine alternata          */
#define GADGHNONE     0x0003  /* Non evidenziare                           */
/*
** Sono sono usate immagini (ON) o bordi (OFF) per il controllo ?
*/
#define GADGIMAGE     0x0004
/*
** Dove si trova l'origine delle coordinate ?
*/
#define GRELBOTTOM    0x0008  /* Ordinate dal fondo (ON) o dalla cima (OFF) */
#define GRELRIGHT     0x0010  /* Ascisse da destra (ON) o da sinistra (OFF) */
/*
** Dimensioni relative (ON) od assolute (OFF) ?
*/
#define GRELWIDTH     0x0020  /* Larghezza relativa (ON) od assoluta (OFF)  */
#define GRELHEIGHT    0x0040  /* Altezza relativa (ON) od assoluta (OFF)    */
/*
** Il controllo è stato selezionato (ON) o no (OFF) ?
** (Inizializzato dal programma e modificato da Intuition)
*/
#define SELECTED      0x0080
/*
** Il controllo è disabilitato (ON) od attivo (OFF) ?
** (Inizializzato dal programma e modificato da Intuition)
*/
#define GADGDISABLED  0x0100
```

Figura 6
Attributi utilizzabili nel campo Flags.

NextGadget

Come al solito è possibile legare più strutture a formare una catena. Questo è appunto il puntatore alla struttura successiva nella lista. Ovviamente è **NULL** se questa è l'ultima o l'unica struttura della catena.

LeftEdge

Posizione orizzontale del bordo sinistro dell'area di selezione del controllo, misurata come spiegato più avanti.

TopEdge

Posizione verticale del bordo superiore dell'area di selezione del controllo, misurata come spiegato più avanti.

Width

Larghezza dell'area di selezione del controllo, misurata come spiegato più avanti.

Height

Altezza dell'area di selezione del controllo, misurata come spiegato più avanti.

Flags

Serve a specificare le caratteristiche statiche del controllo. La lista completa è fornita in figura 6.

Activation

Serve a specificare le caratteristiche dinamiche del controllo. La lista completa è fornita in figura 7.

GadgetType

Serve a definire il tipo di controllo. Questa lista è fornita in figura 8.

GadgetRender

È l'analogo del campo **ItemFill** della struttura **MenuItem** descritta un po' di tempo fa in questa stessa rubrica. Punta alla struttura **Image** o **Border** che determina l'aspetto grafico del controllo. Di quale delle due strutture si tratta dipende da come si è impostato il campo **Flags**, come vedremo. Se non è prevista alcuna immagine o bordo, il campo va impostato a **NULL.**

SelectRender

È l'analogo del campo **SelectFill** della struttura **MenuItem**. Punta alla struttura **Image** o **Border** che determina l'aspetto grafico del controllo quando è selezionato. Di quale delle due strutture si tratta, dipende da come si è impostato il campo **Flags.** Se non è prevista alcuna immagine o bordo, il campo va impostato a **NULL.**

GadgetText

Punta ad una struttura **IntuiText** contenente un testo che si desidera far visualizzare da Intuition accanto al controllo. I campi **LeftEdge** e **TopEdge** della struttura **IntuiText** vanno misurati relativamente all'angolo sinistro superiore dell'area di selezione del controllo. Ovviamente anche questo campo può essere nullo.

MutualExclude

Per il momento questo campo è ignorato da Intuition. Esso ha lo scopo di fornire ai pulsanti un meccanismo automatico di mutua esclusione analogo a quello utilizzato per le voci di un menu. Vedremo in una delle prossime puntate come sia comunque possibile implementare un tale meccanismo nonostan-

```
/* ********************************************************************** **
** Attributi di carattere generale                                        **
** ********************************************************************** */
/*
** Informa Intuition che il programma è interessato a sapere...
*/
#define RELVERIFY       0x0001  /* ...quando il controllo è rilasciato      */
#define GADGIMMEDIATE   0x0002  /* ...quando il controllo è selezionato     */
#define FOLLOWMOUSE     0x0008  /* ...movimenti del puntatore del mouse     */
/*
** Posiziona il controllo...
*/
#define RIGHTBORDER     0x0010  /* ...nel bordo di destra                   */
#define LEFTBORDER      0x0020  /* ...nel bordo di sinistra                 */
#define TOPBORDER       0x0040  /* ...nel bordo superiore                   */
#define BOTTOMBORDER    0x0080  /* ...nel bordo inferiore                   */
/*
** Attributi vari
*/
#define ENDGADGET       0x0004  /* Controllo di chiusura dei quadri         */
#define TOGGLESELECT    0x0100  /* Il controllo è a selezione alternata     */

/* ********************************************************************** **
** Attributi per i CAMPI DI INGRESSO [String Gadgets]                     **
** ********************************************************************** */
#define STRINGCENTER    0x0200  /* Testo centrato                           */
#define STRINGRIGHT     0x0400  /* Testo allineato a destra                 */
#define LONGINT         0x0800  /* Campo di ingresso di tipo numerico       */
#define ALTKEYMAP       0x1000  /* Usa una mappa dei caratteri alternata    */

/* ********************************************************************** **
** Attributi per i PULSANTI [Boolean Gadgets]                             **
** ********************************************************************** */
#define BOOLEXTEND      0x2000  /* Pulsante con ulteriori informazioni      */
```

Figura 7 - Attributi utilizzabili nel campo Activation.

```
/* ********************************************************************** **
** Tipi di controlli                                                      **
** ********************************************************************** */
/*
** Attributi discriminanti
*/
#define GADGETTYPE  0xFC00 /* Tutti i bit usati dagli attributi di tipo     */
#define SYSGADGET   0x8000 /* Controllo di sistema (ON), di programma (OFF) */
#define SCRGADGET   0x4000 /* Controllo di schermo (ON), di finestra (OFF)  */
/*
** Controlli di sistema   ** USATI INTERNAMENTE DA INTUITION (non usare)   **
*/
#define SIZING      0x0010 /* Ridimensionamento della finestra              */
#define WDRAGGING   0x0020 /* Spostamento della finestra nello schermo      */
#define SDRAGGING   0x0030 /* Scorrimento verticale dello schermo           */
#define WUPFRONT    0x0040 /* Porta la finestra di fronte a tutte           */
#define SUPFRONT    0x0050 /* Porta lo schermo di fronte a tutti            */
#define WDOWNBACK   0x0060 /* Porta la finestra dietro a tutte              */
#define SDOWNBACK   0x0070 /* Porta lo schermo dietro a tutti               */
#define CLOSE       0x0080 /* Chiusura (finestra o schermo)                 */
/*
** Controlli di programma ** SOLO QUESTI VANNO USATI NEL CAMPO GadgetType  **
*/
#define BOOLGADGET  0x0001 /* Pulsante                                      */
#define GADGET0002  0x0002 /* (non documentato)                             */
#define PROPGADGET  0x0003 /* Controllo Proporzionale                       */
#define STRGADGET   0x0004 /* Campo di immissione/emissione dati            */
#define GZZGADGET   0x2000 /* Controllo per finestre GIMMEZEROZERO (ON)     */
#define REQGADGET   0x1000 /* Controllo per quadri (ON), per finestre (OFF) */
```

Figura 8 - Attributi utilizzabili nel campo GadgetType.

te questa funzione non sia ancora stata implementata dalla Commodore.

SpecialInfo

Di questo campo parleremo in dettaglio nelle prossime puntate. Esso punta ad una serie di estensioni differenziate della struttura **Gadget**, che servono a riportare quelle informazioni che sono specifiche e necessarie ad i vari tipi di controlli.

GadgetID

Serve ad identificare il controllo in modo da poter essere successivamente utilizzata per determinare quale controllo è stato selezionato. Non ha alcun interesse per Intuition. È fondamentalmente una variabile messa a disposizione del programmatore ed ha un utilizzo analogo a quello dei vari **MenuNumber, ItemNumber** e **SubNumber** utilizzati nella gestione degli eventi da menu.

UserData

Anche questo campo è del tutto ignorato da Intuition. In pratica viene messo a disposizione del programmatore per permettergli di associare ad uno specifico controllo qualunque tipo di dato che possa essere utile nella gestione dell'evento. Ad esempio, il numero di volte che un utente ha selezionato un centro controllo, oppure la lista delle ultime dieci stringhe che sono stati immesse in un certo campo di introduzione dati [*entry field*].

La posizione di un controllo, altro non è che la posizione della sua area di selezione la quale, di solito, ha la forma di un rettangolo. L'unica eccezione si ha con i cosiddetti *pulsanti con maschera*, ma anche in questo caso possiamo sempre pensare la maschera iscritta in un rettangolo di dimensioni date. La posizione dell'origine dell'area di selezione (che nella convenzione Amiga è per definizione l'angolo superiore sinistro), può essere definita rispetto ad uno qualsiasi degli angoli del contenitore impostando opportunamente il campo **Flags** secondo lo schema riportato in figura 2. In pratica i due attributi **GRELBOTTOM** e **GRELRIGHT** indicano se la posizione debba essere misurata o meno rispettivamente a partire dal bordo inferiore e da quello destro del contenitore. Fate attenzione però! Se uno od entrambi questi attributi sono utilizzati i corrispondenti valori di **TopEdge** e **LeftEdge** devono essere *negativi*.

Analogamente, i due attributi **GRELWIDTH** e **GRELHEIGHT** indicano che i valori dei campi **Width** ed **Height** non rappresentano le dimensioni assolute del controllo, bensì la differenza tra le dimensioni del controllo e quelle del contenitore (vedi figura 3). Questo fa sì che Intuition vari le dimensioni del controllo al variare di quelle del contenitore (ad esempio qualora l'utente allarghi o restringa la finestra che contiene il controllo). Anche in questo caso i valori dei due campi sono negativi. Anche qui fate attenzione: a variare è solo l'area di selezione, non l'immagine od il bordo che la veste. A quelli ci dovete pensare voi. Un esempio di controllo che si allarga insieme alla finestra è la barra di spostamento, quella che in genere contiene il titolo della finestra. Due esempi sono riportati in figura 4 e figura 5.

Gli altri valori impostabili in **Flags** sono relativi alla selezione del controllo ed alla sua evidenziazione in tale stato. Li vedremo quando parleremo di bottoni, la prossima puntata. Analogamente vedremo il significato dei vari valori utilizzabili in **Activation** e **GadgetType** man mano che analizzeremo i vari tipi di controlli. Chiudiamo qui questa puntata dicendo che, nel caso il controllo vada posizionato nel bordo di una finestra GZZ, cioè nella mappa di bit relativa alla finestra esterna, è necessario impostare il valore **GZZGADGET** in **GadgetType** (ricordo che di questo particolare tipo di finestre abbiamo parlato nella 9ª puntata pubblicata nel numero 82 di MC del febbrario '89).

Conclusione

Dalla prossima puntata incominceremo a vedere in dettaglio i singoli tipi di controlli.

Man mano che andremo avanti completeremo la descrizione dei valori che i vari campi delle strutture utilizzate possono prendere. Alla fine riassumeremo in una tabella quali campi e quali valori sono significativi per i vari tipi di controlli.

Fatto questo analizzeremo come si gestiscono gli eventi IDCMP generati dai controlli, in modo analogo a quanto già fatto per i menu. Cercheremo di utilizzare la struttura dello scheletro costruito nelle varie puntate dedicate ad i

La scheda tecnica

Anche questo mese ecco cinque comandi dell'AmigaDOS 1.3 a partire da **JOIN**.

LEGENDA	
<parametro>	parametro da specificare
[<opzione>]	parametro opzionale
{<opz-rip>}	parametro opzionale che può essere ripetuto n volte
...	serie che può essere continuata
|	separatore per una lista di opzioni di cui una almeno VA specificata
/A	indica che il parametro DEVE essere specificato
/K	indica che quella determinata parola chiave VA specificata se si vuole usare l'opzione ad essa associata
/S	indica una parola chiave da specificare per attivare l'operazione ad essa associata

Comando:	PATH
Formato:	PATH [SHOW] [ADD <indirizzario>...] [RESET] [QUIET]
Sintassi:	PATH "SHOW/S,ADD,RESET/S,QUIET/S"
Scopo:	Gestisce la lista di ricerca degli indirizzari per i comandi
Specifiche:	Quando un comando viene lanciato da CLI o SHELL, il sistema lo cerca prima nell'indirizzario corrente, poi in SYS:C (o C:), ed infine in tutti quelli specificati dal comando PATH, nell'ordine fornito. PATH da solo (o con SHOW) visualizza il cammino attualmente definito per questa ricerca. ADD invece permette di aggiungere fino a 10 nuovi indirizzari al cammino corrente. RESET permette invece di ridefinire la sequenza completa ad eccezione dell'indirizzario corrente e di SYS:C che sono sempre i primi due elementi della lista. QUIET è utilizzato in congiunzione con SHOW per evitare che, nel caso che uno o più volumi relativi al cammino selezionato non sia disponibile [unmounted], una richiesta di montare il volume appaia sullo schemo [requester]. ADD può anche essere posto in fondo alla lista degli indirizzari.
Esempio:	PATH ADD rad:c rad:s rad:system sys:util

Comando:	NEWCLI
Formato:	NEWCLI [<specifiche finestra>] [FROM <nome file>]
Sintassi:	NEWCLI "WINDOW,FROM/K"
Scopo:	Apre una nuova sessione CLI in una nuova finestra
Specifiche:	Quando la nuova sessione interattiva è aperta, viene automaticamente eseguito il file s:CLI-Startup, se esiste, a meno che non sia stato specificato un file differente con l'opzione FROM. La sessione CLI pura, al contrario di quella SHELL, ha una gestione minimale nella redazione della linea di comando.
Esempio:	NEWCLI con:0/10/640/240/FinestraCLI FROM s:finestra.start

Comando:	PROMPT
Formato:	PROMPT <simbolo>
Sintassi:	PROMPT "PROMPT"
Scopo:	Per modificare il simbolo di "console pronta"
Specifiche:	Mentre per una finestra CLI il simbolo può contenere solo la variabile %N, che verrà sostituita dal numero del CLI in cui si sta operando, in una finestra SHELL è anche disponibile la variabile %S, che viene sostituita in modo dinamico con il nome dell'indirizzario corrente.
Esempio:	PROMPT $e[33m[%N] $e[31m dove, per ragioni tipografiche, $e rappresenta il valore esadecimale 0x1B, a cui non corrisponde un carattere stampabile

Comando:	NEWSHELL
Formato:	NEWSHELL [<specifiche finestra>] [FROM <nome file>]
Sintassi:	NEWSHELL "WINDOW,FROM/K"
Scopo:	Apre una nuova sessione SHELL in una finestra di tipo NEWCON:
Specifiche:	Grazie al gestore di console L:NewCon-Handler, è possibile montare una nuova console logica chiamata NEWCON: con maggiori capacità della vecchia CON:. In particolare NEWCON: permette una migliore gestione dell'editing di linea, e fornisce un buffer storico dei comandi lanciati molto utile qualora sia necessario lanciare lo stesso comando o comandi simili più volte. Il comando NEWSHELL serve a far partire appunto una nuova sessione interattiva basata su NEWCON: e su una SHELL che in aggiunta permette di definire abbreviazioni e sinonimi [aliases] per i comandi più usati. Purtroppo il numero di sinonimi definibili sono fortemente limitati dal fatto che la memoria prevista per mantenere tali informazioni è alquanto limitata. Ovviamente NEWSHELL usa NEWCON: solo se questa è stata precedentemente montata con il comando MOUNT NEWCON:, altrimenti utilizza la vecchia CON:. Similmente, lo SHELL è utilizzato se e solo se il file Shell-Seg è residente al momento dell'invocazione del comando, cosa solitamente effettuata nella Startup-Sequence con l'istruzione 1> resident CLI L:Shell-Seg SYSTEM pure add Se non specificato altrimenti tramite l'opzione FROM, il profilo di partenza del comando NEWSHELL è s:Shell-Startup.
Esempio:	NEWSHELL newcon:0\10\640\240\Conchiglia FROM s:conchiglia.vai!

Comando:	PROTECT
Formato:	PROTECT [FILE] <nome file> [FLAGS] [+|-]<attributi> [ADD] [SUB]
Sintassi:	PROTECT "FILE/A,FLAGS,ADD/S,SUB/S"
Scopo:	Modifica gli attributi di un file
Specifiche:	Il comando 1.3, in aggiunta agli attributi già disponibili nella versione 1.2, permette ora di modificare anche P pure — identifica un comando come "rientrante" S script — rende direttamente eseguibile una macro di sistema A archive — indica che il file è stato archiviato Per una spiegazione sui comandi rientranti, vedere la scheda relativa al comando RESIDENT. Le opzioni ADD e SUB (od i corrispondenti prefissi + e -) possono essere utilizzate per attivare o cancellare uno o più specifici attributi, senza modificare gli altri.
Esempio:	PROTECT s:pcmount +s

Il comando **RESIDENT** *sarà descritto nella prossima puntata.*

menu, ma non ci legheremo ad essa più del necessario. Vedremo quindi quando conviene definire un controllo in modo statico, e quando allocare memoria in modo dinamico. Cercheremo di definire un approccio abbastanza generale che ottimizzi il rapporto tra prestazioni e dimensioni del codice. Vedremo infine come rendere più leggibile e facilmente mantenibile un codice nel quale si debba definire un elevato numero di controlli.

Gli argomenti sono tanti, e forse non sarà possibile analizzarli esaustivamente tutti.

Quindi, se avete domande, problemi, od ancora meglio consigli ed esempi, mandatemeli. Lo scopo di questa rubrica infatti consiste nel dare qualcosa di più di quello che si può capire leggendo i *ROM Kernel Manual* o qualche libro sulla programmazione dell'Amiga. Qualunque contributo originale ed, ovviamente, verificato, è il benvenuto.

Casella Postale

Ancora su FF ed EVAL

Egregio Dott. Dario de Judicibus,
Caro Amigo, ho appena finito di leggere una delle pagine più «appetitose» di MC di settembre (n. 99): la 25ª puntata del suo «Programmare in C su Amiga» e con essa la Scheda Tecnica dedicata all'AmigaDOS.

Sono rimasto piuttosto colpito dall'affermazione che il comando FF <font> sembra non funzionare, visto che perfino sul mio manuale in olandese (!) si accenna alla possibilità di usare tale opzione (sennò, d'altra parte, perché non chiamarlo piuttosto FT come FastText?).

Detto fatto, mi sono messo alla caccia e ho trovato che effettivamente i font in dotazione all'AmigaDOS 1.3 non sono utilizzabili al proposito, avendo verificato, come peraltro afferma lo stesso FF chiamato in causa, che sono font Proportional. Gli unici con cui sono riuscito a ottenere il cambio di font sono... tutti i font 8x8 in mio possesso, ovvero: un tal Pearl.font (8), di cui non ricordo la provenienza, e che per chiarezza tipografica preferisco sommamente al Topaz.font standard il Siesta (8, 9, 11), proveniente dal disco del Risiko (sì, il noto boardgame!) e, of course, Topaz.font.

Precisazione: il mio sistema ha KickStart V.34.5 e WorkBench V.34.28. La versione di FF in mio possesso è la 1.31.

E uno.

Inoltre, anche quanto affermato nella Nota non è perfettamente vero. Effettivamente, << e >> si sono ridotti a < e > metre le operazioni mod, xor e eqv sono presenti, ma come m x e (sempre minuscole). Ovvio no? C'è però da riscontrare un piccolo inconveniente: dato che il parser di eval sembra tenere in poco conto gli spazi, per cui EVAL 8 7 dà 87, se si vuole operare in hex con EQV bisogna fare attenzione, infatti

EVAL 0x7 e 0x8 diventa (tradotta)
EVAL 0x7e 0x8 per cui, trascurando il 2° operando

perché non preceduto da un operatore, fornisce 126 (=0x7e) invece che il corretto -16 (=0xFF...F0). Bisognerà allora sfruttare le parentesi e scrivere:

EVAL (0x7) e 0x8

Ah, mondo birbone!
Amighevoli saluti dal Vostro fedelissimo
Massimo Gais - S. Antimo (NA)

Mentre i comandi più classici dell'AmigaDOS hanno un certo grado di affidabilità, è evidente che un certo numero di programmi di utilità fatti passare come comandi del sistema operativo soffrono di una certa instabilità.

Tali comandi si riconoscono facilmente dagli altri, in quanto non seguono quelle regole di sintassi che, pur mancando di quel grado di standardizzazione che ci si aspetterebbe da un sistema serio, caratterizzano comandi quali **dir, list, cd** e simili. In particolare la regola del punto interrogativo per avere la sintassi completa non sempre è disponibile.

Anche da questo punto di vista **ARP** ha qualcosa da insegnare agli sviluppatori Commodore.

Il risultato è che spesso i risultati ottenuti nell'utilizzo di questi programmi sono imprevedibili, e variano da versione a versione.

E bene ha fatto il Sig. Gais ad indicare la versione del suo **FF**, e cioè la 1.31, perché questo spiega probabilmente l'arcano.

La versione in mio possesso è quella originalmente distribuita con il sistema 1.2, e cioè la 1.1, evidentemente precedente a quella del Sig. Gais. Il KickStart è lo stesso (34.5), ma il WorkBench è il 34.27.

Numeri a parte, sebbene la Commodore abbia dimostrato spesso l'intelligenza di sfruttare al massimo quanto sviluppato ed ideato nel mondo PD per poi inserirlo nelle versioni successive del suo sistema (vedi ARexx), spesso tale operazione è stata effettuata con un po' di superficialità, rilasciando comandi e programmi di scarsa qualità, anche se, per nostra fortuna, di secondaria importanza e comunque facilmente sostituibili con software PD.

Ed il secondo comando di cui il Sig. Gais parla ne è un altro esempio. Per questo è stato da me indicato nella scheda come inaffidabile.

Poco importa, anche se certamente si tratta di una informazione utile, che le funzioni **mod, xor** ed **eqv** sono effettivamente disponibili nella forma abbreviata, contrariamente a quanto afferma il manuale.

Resta il fatto che il *piccolo inconveniente* che porta ad ignorare gli spazi, ed a volte qualcos'altro, rende il tutto alquanto poco usabile.

Ho detto *qualcos'altro* perché provando l'esempio riportato nella lettera, il risultato da me ottenuto è leggermente diverso. Da imputare anche questo a versioni differenti? Non me ne stupirei...

Questi sono i risultati:
EVAL 0x7 0x8 dà 120 cioè 0x78
EVAL 0x7 e 0x8 dà 2024 cioè 0x7e8

In pratica non solo il comando ignora gli spazi, ma non trascura proprio per niente il secondo operando, come affermato nella lettera.

Lo attacca invece al primo, operatore compreso, se presente. Il tutto nonostante il prefisso **0x** ad indicare valori in esadecimale.

Un consiglio? Scrivetevi da soli un comando di calcolo in linea più affidabile.

Si tratta di un ottimo esercizio di C, e di un oggetto estremamente utile e per di più, se si fa un piccolo sforzo, portabile su qualunque sistema. Chi poi avesse l'ARexx, beh, utilizzando la funzione **Interpret** il tutto si riduce ad un paio di linee di codice... Provare per credere.

MC