# Programmare in C su Amiga (35)

*di Dario de Judicibus (MC2120 su MC-Link)*

*Campi stringa e campi numerici. Ovvero: l'immissione dei dati nel programma. Con questa puntata iniziamo a vedere un nuovo tipo di controllo che Intuition ci mette a disposizione. Inizia inoltre una nuova rubrica che prenderà il nome di "Novità" nel mondo Amiga*

## Introduzione

Abbiamo visto nelle scorse puntate come si possono sfruttare al massimo i controlli a pulsante, anche al di là di quelli che sono i controlli «classici», estendendo il tipo *pulsante* con due nuovi tipi: i pulsanti a rilascio incrociato, e quelli a rotazione.

A partire da questa puntata, incominceremo a vedere altri tipi di controlli, e precisamente i campi stringa [*string gadget*], ed i campi numerici [*integer gadget*].

Questi controlli ci permetteranno di aggiungere un altro importante elemento alla nostra interfaccia interattiva, e cioè la possibilità di immettere dati che il programma poi provvederà ad elaborare.

I pulsanti, infatti, ci permettevano solo di effettuare una serie di scelte, o di attivare una o più operazioni, entrambe predefinite dal programma. Con i campi, viene fornita all'utente, una maggiore flessibilità nell'interazione con il programma.

## I campi

Un campo permette all'utente di fornire informazioni al programma sotto forma di stringhe di caratteri. Se vogliamo, un campo rappresenta in un'interfaccia grafica, quello che un parametro rappresenta in una interfaccia a comandi. Viceversa, i pulsanti potrebbero essere paragonati alle opzioni [*switch*] di un comando, in quanto la scelta dell'utente ricade su un insieme di valori o possibilità prestabilite dallo sviluppatore del programma.

Questo è molto importante da capire, in quanto una scelta opportuna del controllo da utilizzare in una interfaccia interattiva è fondamentale ai fini dell'*usabilità* del programma, e di conseguenza al favore che gli utenti finiranno per dare o meno a tale programma. Un semplice esempio, tanto per rimanere sul concreto. Se ad un certo punto del mio programma, ho bisogno di chiedere all'utente il nome di un file da aprire, dovrò aprire un quadro per permettere a tale utente di selezionare il file.

Il quadro più piccolo che posso pensare, è formato da un solo campo di ingresso, senza alcun pulsante. L'utente scrive nel campo il nome del file, e quando preme **invio** il quadro scompare ed il valore del campo viene acquisito dal programma.

Una soluzione come questa presenta molti inconvenienti.

- Innanzi tutto non si dice all'utente cosa ci si aspetta da lui. Non una riga di testo, non una spiegazione. Solo il quadro nudo e crudo.
- Quindi si costringe l'utente a ricordare dove si trovava il file in questione, cioè in quale volume ed indirizzario.
- Terzo, si costringe l'utente a specificare il nome del file *per intero*, cioè completo del nome del volume e del cammino completo degli indirizzari.
- Quarto, non si dà all'utente la possibilità di poter cancellare la richiesta di apertura del file.

Vediamo allora come risolvere il problema.

Innanzi tutto aggiungiamo al quadro un testo di spiegazione, che dica all'utente cosa deve fare o, se il contesto è abbastanza chiaro, un buon titolo, ed eventualmente un pulsante di **Guida** [*Help*] per gli utenti meno esperti.

In secondo luogo, aggiungiamo un meccanismo di navigazione, basato su liste selezionabili, che ci permetta di muoverci nel *filing system* e di trovare il

```
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              */
  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.*

```
struct StringInfo
{
 /*
 **  I seguenti campi sono inizializzati dal programma
 */
  UBYTE         *Buffer     ; /* Area dati del campo                          */
  UBYTE         *UndoBuffer ; /* Area di salvataggio del campo                */
  SHORT          BufferPos  ; /* Posizione del cursore nell'area dati         */
  SHORT          MaxChars   ; /* Numero massimo di caratteri incluso \0       */
  SHORT          DispPos    ; /* Posizione del primo carattere visibile       */
 /*
 **  I seguenti campi sono mantenuti da Intuition
 */
  SHORT          UndoPos    ; /* Posizione del cursore nell'area salvataggio */
  SHORT          NumChars   ; /* Numero di caratteri attualmente immessi     */
  SHORT          DispCount  ; /* Numero di caratteri visibili nel contenitore*/
  SHORT          CLeft,CTop ; /* RISERVATO: posizione del contenitore        */
  struct Layer  *LayerPtr   ; /* RISERVATO: piano contenente il controllo    */
  LONG           LongInt    ; /* Intero immesso (per campi numerici)         */
 /*
 **  Il seguente campo è fornito dal programma
 */
  struct KeyMap *AltKeyMap  ; /* Tastiera alternativa (vedi testo)           */
};
```

*Figura 2 - La struttura StringInfo*

```
/* ********************************************************************** **
**  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 GADGHNONE     0x0003  /* Non evidenziare                           */
/*
**  Vanno 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 3 - Attributi utilizzabili nel campo Flags applicabili ai campi.* ▶

file che ci interessa aprire. Vedremo nelle prossime puntate come si scrive un *File Requester*.

Terzo, il programma deve essere abbastanza intelligente da ricordare l'indirizzario da cui è stato fatto partire, o quello dell'ultima richiesta di apertura di un file. Questo nell'eventualità, abbastanza comune, che il file si trovi nello stesso indirizzario del programma, o che l'utente stia operando sempre sullo stesso indirizzario di lavoro [*work directory*].

Infine, non deve essere la pressione del tasto di **invio** e far chiudere il quadro, ma la selezione di un pulsante di conferma [*OK*]. Parimenti, un pulsante per la cancellazione dell'operazione deve essere sempre disponibile [*Cancel*].

E se l'utente *sa già* quale file vuole aprire, e trova più comodo digitarlo direttamente senza navigare qua e là utilizzando le liste scrollabili dei file e degli indirizzari?

Ecco allora che *comunque* il campo di ingresso per il nome del file va messo nel quadro, magari con la possibilità di utilizzarlo anche per inserire un modello [*pattern*] di ricerca da usare in congiunzione con le liste scrollabili, e sincronizzato con la selezione via mouse (cioè, quando l'utente seleziona un file da una lista, questo viene automaticamente ricopiato nel campo di ingresso).

Chiusa questa parentesi su quanto sia importante fare attenzione a tutte le possibili esigenze degli utenti quando si disegna una interfaccia, torniamo ai controlli di tipo campo.

Intuition permette di creare due tipi di campi. Quello per l'ingresso di stringhe di caratteri, che chiameremo d'ora in poi *campo stringa*, e quello per l'ingresso di campi di tipo numerico, per la precisione di interi, come vedremo fra poco. Chiameremo quest'ultimo *campo numerico*.

Al contrario di altri sistemi operativi, Intuition non fa distinzione tra campi di ingresso e campi di uscita. In alcuni sistemi, ad esempio, se un campo è di sola uscita, l'utente non può editare il valore in esso contenuto. In Intuition i campi sono prevalentemente pensati come campi di ingresso dati, anche se nessuno impedisce di utilizzarli per visualizzare valori assegnati da programma. L'utente può tuttavia *sempre* editare il valore nel campo, a meno che il controllo non sia disabilitato, ovviamente.

Vediamo in dettaglio come Intuition tratta i campi.

Un campo è formato da due elementi. Un contenitore, ed un'area dati. L'area dati, come dice il nome, è semplicemente un'area di memoria [*buffer*] di lunghezza sufficiente a contenere il valore associato al campo. Il contenitore è invece una finestra su tale area, che rappresenta la parte del campo che verrà visualizzata sullo schermo.

Supponiamo ad esempio di aver scritto un programma per l'immissione di schede [*record*] in una base dati [*database*]. Può succedere che un elemento della scheda sia più lungo della massima stringa che si può visualizzare in un quadro. Ad esempio, se l'elemento in questione è il titolo di un film della Wertmuller, gli 80 caratteri tipici di uno schermo WorkBench utilizzato il font standard non basteranno sicuramente! Nessun problema. Basta definire il contenitore lungo 80, od anche 60 caratteri, e l'area dati da 256 caratteri (dovrebbero bastare, penso). Quando il cursore avrà raggiunto la fine del contenitore, Intuition si prenderà carico di far scorrere il testo verso sinistra, in modo da permettere l'inserimento dell'intero titolo.

Ovviamente si può anche muovere il cursore indietro per vedere la parte iniziale del testo. Il programma non ha da fare niente. Pensa a tutto Intuition.

Un campo può avere anche un terzo elemento, opzionale, e cioè un'area dati di salvataggio [*undo buffer*]. Tale area deve essere uguale o maggiore come lunghezza a quella dell'area dati associata al campo. Più campi possono condividere la stessa area di salvataggio, a condizione che essa sia lunga quanto la più lunga fra le aree dati associate ed i campi in questione.

L'area di salvataggio permette all'utente di ripristinare il valore precedentemente contenuto nel campo (prima cioè che l'utente lo modificasse), semplicemente premendo il tasto *Amiga* a de-

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

◀ *Figura 4 - Attributi utilizzabili nel campo Activation applicabili ai campi.*

```
/* ********************************************************************** **
** Tipi di controlli applicabili ai campi                                 **
** ********************************************************************** */
/*
** Controlli di programma ** SOLO QUESTI VANNO USATI NEL CAMPO GadgetType  **
*/
#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 5 - Attributi utilizzabili nel campo GadgetType applicabili ai campi.*

```
BOOL ActivateGadget( struct Gadget *, struct Window *, struct Requester * ) ;
```

*Figura 6 - ActivateGadget().*

stra della barra spaziatrice, e contemporaneamente la lettera *Q*. L'area di salvataggio viene aggiornata quando l'utente seleziona il campo con il mouse. A quel punto essa rimane invariata fino alla successiva selezione. Viceversa, l'area dati viene inizializzata dal programma prima che il campo venga visualizzato. Attenzione. La stringa di inizializzazione deve sempre contenere un valore nullo in fondo, cioè **0x00**. Ad esempio, se si vuole inizializzare il campo con una stringa nulla, bisogna mettere il valore **0x00** nel primo byte dell'area dati.

Il programma può accedere al valore immesso dall'utente, o quando questi preme **invio**, oppure quando seleziona un altro controllo od un menu. In entrambi i casi, il campo viene automaticamente deselezionato da Intuition. Al contrario quindi dei pulsanti, che possono essere di tipo automatico, e cioè rimangono selezionati solo fintanto che l'utente li tiene «premuti» con il mouse, o di tipo manuale, mantenendo cioè il loro stato indipendentemente da quello di altri controlli, un campo rimane selezionato fintanto che l'utente ci lavora, e non ci possono essere altri campi «attivi» nello stesso momento. Ecco perché una sola area di salvataggio può essere condivisa da tutti i campi di un quadro o di una finestra.

Il testo in un campo può essere allineato a sinistra, a destra, oppure centrato. Il default è l'allineamento a sinistra.

Il contenitore del campo è in realtà un rettangolo selezionabile [*select box*], e benché possa essere reso utilizzando un'immagine od un bordo, o più semplicemente non associato ad alcun elemento grafico, l'unico tipo di evidenziazione attualmente supportato è quella *a colore complementare*. Non è quindi possibile usare un'immagine od un bordo alternativo, né chiedere ad Intuition che evidenzi il campo selezionato con una cornice automatica [*box drawing*].

L'utente ha la possibilità di usare un certo numero di tasti per editare il campo. Le frecce *cursore sinistro* e *cursore destro* servono appunto a spostare il cursore di un carattere alla volta sul testo contenuto nel campo. Se con tali tasti si usa anche il tasto di **shift**, il cursore si sposta rispettivamente all'inizio ed alla fine del testo. Il tasto **Canc** [*DEL*] cancella il carattere sotto il cursore, mentre lo spazio indietro [*backspace*] cancella quello a sinistra del cursore. Il tasto di **Invio** [*Return*] termina l'immissione dei dati e deseleziona il campo. Se, come già detto, si preme contemporaneamente il tasto *Amiga* destro, e la lettera **Q**, si ripristina il contenuto dell'area di salvataggio. Se invece si preme con il tasto *Amiga* il tasto **X**, si cancella l'area dati, lasciando tuttavia inalterata l'area di salvataggio.

Tutti gli altri tasti o combinazioni, che normalmente servono a visualizzare un carattere, inseriscono quel carattere alla posizione corrente del cursore. Questo in quanto i campi sono sempre di tipo «a inserimento automatico» [*auto-insert*]. Non serve quindi premere a tale scopo il tasto **Ins.**

I campi numerici accettano solo interi con segno, da 32 bit. Intuition verifica che solo i due segni + e −, ed i numeri da **0** a **9** vengono utilizzati. Per il resto valgono le stesse regole di edizione sopra menzionate. A differenza del campo stringa, il campo intero mette a disposizione del programma una variabile nella quale, come vedremo, ritorna il valore dell'intero immesso dall'utente. L'inizializzazione del campo intero, tuttavia, non si ottiene assegnando un intero a tale variabile, ma inizializzando l'area dati con una stringa di caratteri il cui valore fornisca la rappresentazione ASCII del numero intero in questione. La variabile che ritorna il numero immesso dall'utente, infatti, è solo una variabile di servizio che evita al programma di effettuare la conversione da stringa a numero equivalente, cosa a cui pensa Intuition.

## *Le strutture C*

Vediamo ora le strutture C necessarie a definire un campo.

La struttura base resta sempre e comunque la struttura **Gadget**, già descritta nella 28ª puntata (*MCmicrocomputer n. 102*) e riportata, per comodità, in figura 1.

I campi **NextGadget**, **LeftEdge**, **TopEdge**, **Width**, **Height**, **GadgetID** e **UserData** vanno impostati come per tutti gli altri tipi di controlli, a discrezione del programmatore.

Il campo **MutualExclude** non va utilizzato, come al solito.

Il campo **GadgetType** potrà contenere solo alcuni dei valori possibili, come riportato in figura 5, ed in particolare **StrGad-**

**get**, ad indicare che vogliamo definire un campo stringa o numerico.

Il campo **GadgetRender** punterà ad un'immagine, ad un bordo, od a niente, a seconda di come vogliamo visualizzare il contenitore. Anche qui, come già per i pulsanti, non c'è nessun controllo automatico che l'immagine utilizzata copra esattamente l'area di selezione di controllo, che nel caso in questione corrisponde al contenitore. Sta al programmatore garantire tale corrispondenza.

Il campo **SelectRender** non è utilizzato, al momento, per i campi, dato che l'unica tecnica valida di evidenziazione, è quella a complemento di colore.

Il campo **GadgetText** non è solitamente utilizzato per i controlli di tipo campo, ma può viceversa risultare molto interessante. Lo vedremo nella prossima puntata.

Il campo **Flags** potrà contenere solo alcuni dei valori possibili, come riportato in figura 3. Per quello che riguarda la tecnica di evidenziazione, come già detto, non potrà essere utilizzato né **GADGHBOX**, né **GADGHIMAGE**.

Il campo **Activation**, viceversa, oltre ai valori validi per un controllo generico, potrà contenere alcuni valori addizionali, riportati in figura 4. In particolare, **STRINGCENTER** e **STRINGRIGHT**, per modificare l'allineamento del testo nel contenitore;
**LONGINT**,
ad indicare che vogliamo definire un campo numerico;
**ALTKEYMAP**,
nel caso intendessimo utilizzare una tastiera alternativa.

Tutto ciò, però, non basta a definire un campo. Sono necessarie altre informazioni. Per assegnarle, è necessario utilizzare quindi una struttura speciale, il cui puntatore viene assegnato appunto al campo **SpecialInfo** della struttura **Gadget**. Questa nuova struttura si chiama **StringInfo**, e contiene sia campi che vanno assegnati dal programma, sia campi contenenti informazioni varie relative al controllo, e mantenute da Intuition.

### *La struttura StringInfo*

La struttura **StringInfo** serve quindi a definire alcune caratteristiche peculiari dei controlli di tipo campo, non presenti nella struttura base **Gadget**. Questa struttura, riportata in figura 2, ha ben tredici campi.

**Buffer**
è il puntatore all'area dati, la cui allocazione e deallocazione è responsabilità del programma, non di Intuition. Ogni campo ha una ed una sola area dati ad esso associata.

**UndoBuffer**
è il puntatore all'area di salvataggio, che permette di ripristinare il contenuto dell'area dati precedente alle modifiche effettuate dall'utente. Anche l'allocazione e la deallocazione di tale area è responsabilità del programma, ma essa può essere condivisa da più campi, e non va quindi distrutta alla rimozione di uno qualunque dei campi che la utilizza, se questo è il caso.

**BufferPos**
contiene la posizione del cursore nell'area dati. Da notare che tale posizione non coincide necessariamente con la posizione del cursore nel contenitore, dato che questi può contenere meno caratteri dell'area dati, potendo il testo scorrere.

**MaxChars**
definisce il numero massimo di caratteri che l'area dati può contenere, carattere nullo di fine stringa incluso.

**DispPos**
contiene la posizione nell'area dati del primo carattere visibile nel contenitore. La posizione del cursore nel contenitore, quindi, è data da **BufferPos** meno **DispPos**.

**UndoPos**
contiene la posizione del cursore nell'area di salvataggio. Questo campo è gestito da Intuition.

**NumChars**
contiene il numero di caratteri che al momento sono stati immessi nell'area dati. Questo campo è gestito da Intuition.

**DispCount**
contiene il numero di caratteri completamente visibili nel contenitore. Se un carattere è visualizzato solo in parte, non viene preso in considerazione. Questo campo è gestito da Intuition.

**CLeft**
questo campo è riservato ad Intuition, e non va modificato da programma. Esso è relativo alla posizione del contenitore.

**CTop**
questo campo è riservato ad Intuition, e non va modificato da programma. Anch'esso è relativo alla posizione del contenitore.

**LeyerPtr**
contiene il puntatore al piano che contiene il controllo. Questo campo è riservato ad Intuition, e non va *assolutamente* modificato dal programma.

**LongInt**
ha significato solo per i campi numerici, ed è utilizzato da Intuition per riportarvi la conversione dell'area dati di un campo numerico nell'intero corrispondente. Questo campo è gestito da Intuition.

**AltKeyMap**
punta ad un'eventuale tastiera alternativa specificata dal programma.

Una cosa importante, altrimenti potreste avere qualche problema nel lavorare con i campi stringa. Sia la posizione del cursore, che quella del primo carattere visibile, partono da *zero*, non da uno. Questo rende più semplice referenziare il carattere nell'area dati, dato che la convenzione è la stessa degli indici dei vettori, in C. Potrebbe però creare qualche problema nei calcoli, se lo dimentica. Fate attenzione, quindi.

Il campo **AltKeyMap** sarà analizzato in seguito, quando termineremo la serie di articoli dedicati ai controlli, e parleremo un po' di tastiere, mappe dei tasti, *console.device*, ed altro ancora. Per il momento facciamo solo un esempio di possibile utilizzo.

Supponiamo che il nostro programma sia in grado di operare su un dischetto nel

```
; Controllore di schermo (Davide Ficano 17.04.91 Palermo)
; Versione BOOT
;
parto: dc.l $444f5300
       dc.l $4484368c ;Nuovo BootCheckSum
       dc.l $370
       MOVE.L  $4,A6
       CMP.B #$32,$0212(A6)
       BEQ.S   fine
       JSR -120(A6)
       JSR -150(A6)          ;Vedi versione CLI
       MOVE.L #$4E704EF9,0
       JMP $0
fine:  lea dos(pc),a1
       jsr -96(a6)
       tst.l d0
       beq.s err
       movea.l d0,a0
       movea.l $16(a0),a0
       moveq #0,d0
usc:   rts
err:   moveq #-1,d0
       bra.s usc
dos:   dc.b 'dos.library',0
       dc.b "QUESTO NON E' UN VIRUS MA SOLO UN BOOTBLOCK"
       dc.b " CHE CONTROLLA SE LO SCHERMO E' PAL."
       dc.b "CREATO DA DAVIDE FICANO.SE FOSSE UN VIRUS"
arrivo:dc.b " NON METTEREI IL MIO NOME!!"
       dcb.b 1024-(arrivo-parto),0
```

*Figura 7 - Controllore di schermo □ Versione BOOT.*

```
; Controllore di schermo (Davide Ficano 17.04.91 Palermo)
; Versione CLI
;
  MOVE.L $4,A6
  CMP.B #$32,$0212(A6)
  BEQ.S FINE
; Ecco la nuova routine di Reset
  JSR -120(A6)  ;Disabilitiamo le Interrupts
  JSR -150(A6)  ;Andiamo in Stato SuperVisore
  MOVE.L #$4E704EF9,0  ;Vedi Sotto il Significato
  JMP $0
FINE:
  CLR.L b0
  RTS
;Il valore passato in 0 non è un numero magico ma esso è
;composto dagli OPCODES di due Mnemonici che sono
;#$4E70 RESET
;#$4EF9 JMP 0
```

*Figura 8 - Controllore di schermo □ Versione CLI.*

formato *MS-DOS*, magari utilizzando uno dei tanti prodotti che permettono ad Amiga di leggere questo tipo di dischetti. La convenzione FAT per questo tipo di file prevede un nome formato da due parti: una, detta *nome base*, che va da uno ad otto caratteri al massimo, ed una, detta *estensione*, che ha al massimo tre caratteri, e può non esserci. Le due parti sono separate da un punto. In aggiunta, esistono ben precise limitazioni sui caratteri che possono essere utilizzati e precisamente:

primo, il sistema non fa differenza fra caratteri alfabetici in maiuscolo e caratteri minuscoli;
secondo, sono ammessi solo caratteri alfabetici puri (cioè non accentati), i numeri da zero a nove, i cosiddetti caratteri *nazionali* americani (il simbolo del dollaro, della «e» commerciale, e del cancelletto), il simbolo del percento, l'apostrofo dritto e rovescio, le parentesi tonde e graffe (ma non le quadre), il segno meno, l'accento circonflesso, il punto esclamativo, il trattino sottolineatura e l'*at*.

Nella versione 3.3 del DOS, vengono accettati anche altri caratteri, come il simbolo della sterlina o la *cedilla*, mentre le vocali accentate, ad esempio, vengono convertite, se possibile, in vocali maiuscole accentate, altrimenti in vocali maiuscole semplici.

Supponiamo ora che l'utente debba immettere il nome di un file da creare su di un dischetto MS-DOS. Potremmo usare un campo qualunque, leggere il nome immesso nell'area dati, ed effettuare tutta una serie di controlli *per ogni carattere immesso*.

Ma se noi ci carichiamo la tastiera correntemente usata dal nostro utente, e disabilitiamo i caratteri non ammessi, basterà associare questa tastiera a tutti i campi che dovremmo controllare per evitare tutto quel lavoro ogni volta.

# Novità

In questa puntata ho pensato di iniziare una nuova rubrica, che si aggiunge così alla *Scheda Tecnica* ed alla *Casella Postale*. In questa rubrica parleremo di novità nel mondo della programmazione in C su Amiga. Nuove versioni di compilatori, compilatori PD, strumenti di utilità di vario tipo, CASE e via dicendo.

Questa rubrica non sarà presente tutti i mesi, ma si alternerà alle altre per proporre ai lettori sempre nuovi spunti che rompano un pochino la *monotonia*, spero non eccessiva, della trattazione tecnica dei vari argomenti che volta per volta vengono proposti in questa serie di articoli.

Questo mese parleremo dell'ultima versione del compilatore *Lattice C*, o meglio, la prima versione distribuita con il marchio *SAS*, e cioè il *SAS/C 5.10*.

Verso la fine dello scorso anno, il *SAS Institute Inc.*, che ha sede a Cary, in North Carolina, ha acquisito la gestione completa ed il supporto dell'ex-*Lattice C Development System*, ora rinominato *SAS/C Development System for AmigaDOS*. In effetti, la *SAS*, che è una delle più grosse compagnie di software indipendenti, con oltre due milioni di utenti in 88 paesi del mondo, era di fatto la compagnia a cui la *Lattice* aveva subappaltato la maggior parte del codice sviluppato per il compilatore C distribuito con il marchio *Lattice*.

Con tale acquisizione, la *SAS* ha di fatto acquisito anche gli utenti registrati del vecchio *Lattice C*, tra cui il sottoscritto, che si son visti recapitare l'annuncio della nuova versione, ora ufficialmente denominata *SAS/C 5.10*, con una offerta di *upgrade* dalla versione *5.04* per soli $40 più spese postali. Lo stesso annuncio affermava inoltre che d'ora in poi tale versione (e seguenti) sarebbe stata la sola a usufruire del completo supporto tecnico che la *SAS* offre agli utenti registrati.

Il pacchetto di *upgrade* contiene sei dischetti, un certo numero di pagine da aggiungere ad i vecchi manuali del *Lattice C 5.04*, e due nuove etichette che vanno a sostituire quelle infilate nelle tasche laterali dei due raccoglitori ad anelli della vecchia versione. Il grosso della nuova versione, tuttavia, è proprio nei sei dischetti che sostituiscono completamente quelli della versione precedente, peraltro già molto buona rispetto alla oramai ultra-sorpassata e limitata *4.0*.

Innanzi tutto la nuova versione contiene sia i file di inclusione C ed Assembler 1.3 che quelli della nuova versione del sistema operativo 2.0. Peccato che la Commodore non si sia ancora decisa a rilasciare la documentazione della 2.0 anche agli sviluppatori non professionisti, a quelli cioè che, non facendo soldi con i propri programmi, non si possono permettere il lusso di pagare la non eccessiva, ma comunque consistente, iscrizione all'albo degli sviluppatori registrati Amiga, i soli ad aver diritto, previa garanzia di non ridistribuzione, di ricevere tali informazioni.

Il programma di installazione e personalizzazione delle opzioni iniziali, peraltro dotato di una interessante interfaccia *alla 2.0*, permette di decidere quali file caricare (*1.3, 2.0* od entrambi).

| Opzione | Startup |
|---|---|
| default | lib:c.o |
| -t | lib:cback.o |
| -tb | lib:cback.o |
| -tr | lib:cres.o |
| -tc | lib:catch.o |
| -tcr | lib:catchres.o |
| -t=<path> | <path> |

*Figura A - Opzione* **-t**.

Il limite al numero di paramentri per la **#pragma libcall** è stato portato a 14, permettendo così di chiamare qualsiasi funzione delle librerie Amiga con i parametri nei registri.

L'editor LSE ha adesso una interfaccia ARexx, cosa che permette di crearsi le proprie macro di edizione sfruttando la potenza e la semplicità di programmazione dell'ARexx, e di far comunicare l'LSE con altri programmi dotati di interfaccia ARexx. In aggiunta LSE è ora in grado di creare icone per tutti i file, e supporta schermi WorkBench con 8 e 16 colori.

A proposito, anche se l'ARexx è oramai stato incluso nella nuova versione del sistema operativo, chi non può o non vuole montare la 2.0 sulla propria macchina, può ora richiedere la nuova versione dell'ARexx, la 1.15 che sostituisce la vecchia 1.10.

Il SAS/C 5.10 permette inoltre di editare, compilare, ricucire [*link*], e lanciare un programma *tutto* da WorkBench, grazie alla nuova interfaccia WorkBench per chi non ama il CLI. Persino **LMK** può ora essere lanciato da WorkBench. Questo ovviamente senza togliere nulla ai programmatori più orientati a lavorare da CLI, i quali potranno continuare ad usare il SAS/C nel modo «classico».

L'efficienza del compilatore è stata aumentata. **LC1** è stato velocizzato, specialmente nella generazione delle informazioni di controllo introducibili con l'opzione **-d** [**debug**], mentre **LC2** è stato modificato in modo da generare un codice più efficiente. La cosa più impressionante è l'aumento delle prestazioni di **BLINK**. Ho potuto constatare di persona un incremento della velocità di generazione dell'eseguibile fino a *tre* volte quella precedente, anche se il manuale

Attenzione però. Non codificate la tastiera alternativa al vostro programma. Certo, così è anche più semplice, ma il programma funzionerebbe solo con le tastiere di alcuni paesi. E non fate assunzioni neppure sulla posizione dei tasti alfabetici, dato che uno può sempre usare una tastiera Dvorak.

Un'altra possibilità potrebbe essere quella in cui tutti i tasti sono disabilitati tranne quelli numerici ed i simboli di valuta, da utilizzare per campi intesi per l'immissione di somme in varie valute. Come vedete le possibilità sono svariate, a condizione, ovviamente, di saper codificare una tastiera. Ma questo lo vedremo in seguito.

### *ActivateGadget()*

Per finire, due parole sulla funzione **ActivateGadget**, il cui prototipo è riportato in figura 6. Questa funzione permette di attivare un campo da programma, cioè si comporta come se l'utente avesse selezionato il campo in questione. Essa può tuttavia funzionare se l'utente stava nel frattempo lavorando con i menu o spostando il cursore di controllo proporzionale (a potenziometro). Inoltre è necessario che la finestra od il quadro a cui il campo appartiene, siano esse stesse aperte ed attive. Per garantirsi ciò, tuttavia, non basta chiamare la **ActivateGadget()** subito dopo una **OpenWindow()** od una **Request()**, ma bisogna attendere un evento di tipo **ActiveWindow** o **ReqSet**, rispettivamente. Ovviamente bisognerà essersi assicurati di aver detto ad Intuition che si vuole riceverli, quei messaggi.

### *Conclusione*

Bene. Anche per questa volta abbiamo finito. Nella prossima puntata vedremo come si gestisce un campo di tipo stringa ed uno di tipo intero. Buona programmazione! MC

*Figura B*
*Commenti negli stili C e C++.*

```
/*  Questo è un blocco di commenti secondo lo stile C "classico"
**  Posso chiuderlo a capo, in modo da lavorare tranquillo anche in
**  modalità inserimento, ma perdo così una linea. Per giunta non posso
**  fare lo stesso con i commenti di fondo linea, che vanno chiusi sulla
**  stessa linea.
*/

UnFilmDi007(mai,dire,mai); /* Commento di fondo linea C: devo chiuderlo */

//  Questo è un blocco di commenti secondo lo stile C++
//  Posso lavorare in modalità INSERIMENTO senza preoccuparmi della chiusura
//  del commento. E guadagno un paio di caratteri nei commenti di fondo linea.

UnFilmDi007(mai,dire,mai); // Commento di fondo linea C++: non devo chiuderlo
```

riporta prudentemente un fattore *due*. Ovviamente dipende dal programma che si sta sviluppando.

In aggiunta **LC1** ora supporta un nuovo attributo nelle dichiarative: **_aligned**, che permette di allineare l'oggetto dichiarato alla successiva parola lunga [*longword*]. Questo è richiesto da alcune strutture AmigaDos, come quelle relative alle informazioni sui file. Ad esempio.

**struct FileInfoBlock _aligned fib ; /* Due caratteri di sottolineatura */**

L'opzione **-pr** di **LC1** e **LC1B**, utilizzata per generare prototipi, supporta ora anche i tipi definiti con l'istruzione **typedef**.

Inoltre **LC1** converte automaticamente dati dichiarati **near** in dati **far**, se si dichiarano troppi dati del primo tipo, rendendo di fatto impossibile comprimere tutti i dati nei confini ristretti che tale attributo richiede.

Tre nuovi messaggi di attenzione [*warning*] sono stati aggiunti:

- commenti annidati [*nested*] con annidamento disabilitato
- commento sbilanciato
- blocco **#if/#ifdef** sbilanciato.

Le opzioni di compilazione possono ora essere memorizzate in un file od in una variabile d'ambiente [*environment variable*], in modo che non è più necessario doverle fornire ogni volta sulla linea comandi. Ovviamente tale possibilità è utile se usate spesso le stesse opzioni di compilazione e non fate uso dei **makefile**.

Il programma di profili **LPROF** supporta ora anche blocchi multipli di codice [*multiple code hunk*]. Questo vuol dire che non è più necessario usare l'opzione **SMALLCODE** con **BLINK** quando si usa **LPROF**.

Un'altra utile possibilità, è quella che permette di estendere il *buffer* di espansione del preprocessore, permettendo così di superare le limitazioni del valore di *default*. Potremo quindi evitare errori del tipo «*preprocessor buffer overflow*» qualora una macro si espanda oltre una certa lunghezza.

Sempre per quello che riguarda la memoria, è ora possibile specificare durante la compilazione il valore minimo di *stack* richiesto dal programma. Se il programma così generato viene lanciato con uno *stack* inferiore, sarà lo stesso programma a ridefinirne il valore prima di iniziare l'elaborazione vera e propria.

Se si usa il comando **1c -L**, è ora possibile specificare quale file di *startup* utilizzare fra quelli messi a disposizione dal compilatore (**c.o, cback.o**, e simili), utilizzando l'opzione **-t**, come riportato in figura A.

Per quello che riguarda l'Assembler, esso ora accetta anche la direttiva **EQUR**.

Un'interessante caratteristica del SAS/C 5.10, è che ora **LC1** riconosce i commenti secondo lo stile C++, e permette di mescolarli con quelli classici del C. Questa nuova caratteristica, se da una parte va utilizzata con cura, riducendo al portabilità del codice fra compilatori e sistemi differenti, rende molto più immediato aggiungere e modificare i commenti a fondo linea. Per chi non conoscesse lo stile C++, in figura B ho riportato un esempio.

Ovviamente sono stati fissati anche numerosi bachi minori, ed aggiunte nuove funzioni e macro.

Ora il SAS/C supporta molte varietà della macro **offsetof**, incluse quelle GNU e XWindows. La più semplice utilizzata è la seguente:

**#define offsetof(tipe,name) ((long)&(((type *)0)->name))**

Inoltre sono state aggiunte le seguenti funzioni UNIX:

**opendir**
**readdir**
**seekdir**
**telldir**
**rewinddir**
**closedir**
**stat**
**isatty**

E questo è quanto. Ah, dimenticavo di dire che ho richiesto la nuova versione del SAS/C il 20 di dicembre; il pacco è stato spedito dagli USA il 2 gennaio, ed è arrivato, grazie alle Poste Italiane, a metà febbraio, tanto per cambiare. Forse l'hanno mandato per nave, per paura che venisse dirottato da qualche terrorista fedele a Saddam Hussein... MC