Programmare in C su Amiga (36)

di Dario de Judicibus (MC2120 su MC-Link)

Seconda puntata dedicata ai campi.
Vedremo come si crea un campo, come lo si visualizza sullo schermo, e come lo si rimuove dallo schermo e dalla memoria

Introduzione

Nella scorsa puntata abbiamo incominciato a parlare del secondo controllo base dei tre che Intuition mette a disposizione dei programmatori: il campo.

Abbiamo detto come esistano due tipi di campi: i campi *stringa*, ed i campi *numerici*. Di fatto essi sono praticamente identici, salvo per il fatto che, mentre nei campi stringa è possibile introdurre qualunque tipo di carattere, Intuition impedisce all'utente di introdurre in campo numerico qualsiasi carattere ad eccezione delle cifre da *zero* a *nove*, e dei simboli *più* e *meno* in prima posizione. In effetti, più che di campi numerici, sarebbe più corretto parlare di campi *interi con segno*.

Abbiamo anche detto come Intuition non distingue tra campi di *ingresso*, *uscita*, od *ingresso & uscita*. Sta al programma occuparsi della cosa. Viceversa il sistema solleva il programmatore dal gestire direttamente le operazioni di edizione del contenuto del campo effettuate dall'utente del programma. In questa puntata vedremo come si crea un campo, lo si visualizza, e lo si rimuove dallo schermo. Nella prossima puntata vedremo come si gestisce il campo da programma e come si acquisisce il valore introdotto nel campo dall'utente, ai fini di una successiva elaborazione.

Le funzioni base

Come già per i pulsanti, abbiamo definito tre funzioni. Una per creare dinamicamente le strutture necessarie per la definizione del campo **(CreateField())**, una per visualizzare il campo nel contenitore **(DisplayField())**, ed una per rimuovere il campo dal contenitore e dalla memoria **(DeleteField())**.

A queste si aggiungono due macro utilizzate per ottenere rispettivamente dai campi stringa e da quelli numerici il valore introdotto dall'utente **(StringFieldValue()** e **NumericFieldValue())**.

Vediamo in dettaglio le funzioni in questione. Le macro le vedremo nella prossima puntata.

CreateField()

La procedura principale, come al solito, è quella di creazione. Prima di entrare in dettaglio nella logica della funzione, tuttavia, vediamo come abbiamo deciso di disegnare il controllo. Anche in questo caso, infatti, si è cercato di dare al puro controllo Amiga un certo valore aggiunto, rendendolo più ricco di caratteristiche, e più flessibile da usare.

Diciamo subito che ad un campo che si rispetti va in genere associato un testo, il quale spieghi all'utente qual è lo

Figura 2 ▶ Posizionamento del testo di un campo.

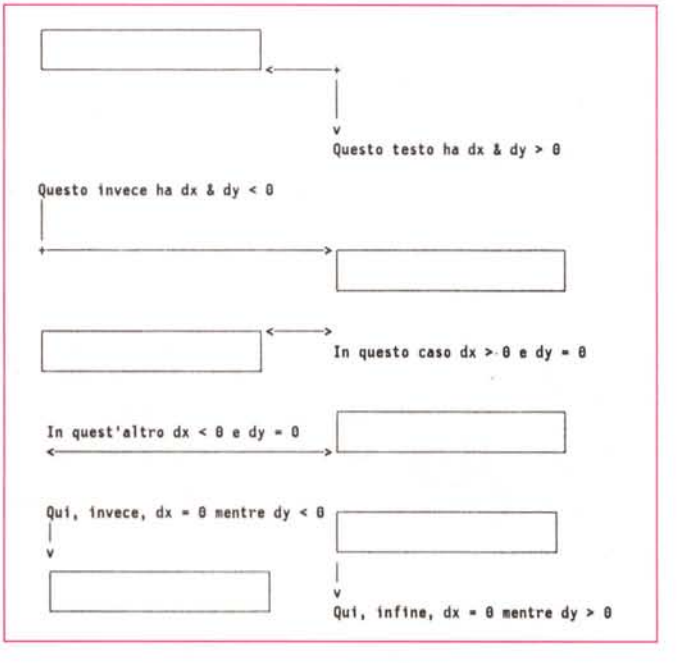
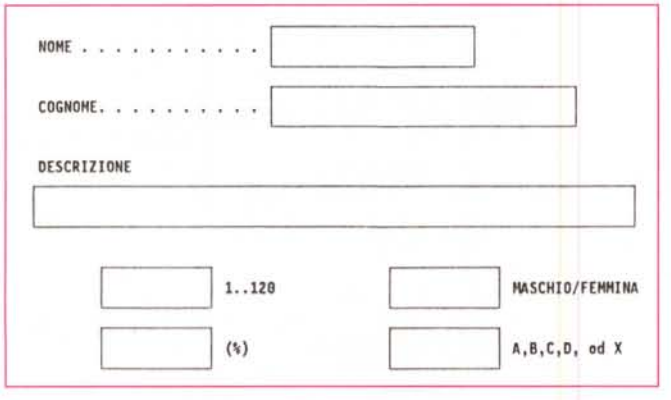


Figura 1 Esempi di campi. ▼



```
/*************************************************************************
** CreateField()                FUNZIONE              Versione 1.00      **
**                                                                       **
** Funzione fornita:  crea dinamicamente un campo di I/O                 **
**                                                                       **
** Dati in ingresso:  id          identificativo del campo               **
**                    container   contenitore (finestra e/o quadro)      **
**                    txt         testo del campo                        **
**                    defval      valore di default del campo            **
**                    fldlen      lunghezza vera del campo               **
**                    boxlen      lunghezza visibile del campo           **
**                    x, y        posizione del campo nel contenitore    **
**                    dx, dy      posizione del testo rispetto al campo  **
**                    class       classe del campo (tipo campo)          **
**                                                                       **
** Dati in uscita:    Gdg         puntatore al campo                     **
**                                                                       **
** Dati globali:      User        memorizza in Size la memoria utilizzata **
**                                                                       **
** Eventi emessi:     GADGETUP    sempre                                 **
**                                                                       **
*************************************************************************/

IGDG *CreateField(id,container,txt,defval,fldlen,boxlen,x,y,dx,dy,class)
  USHORT id ;
  ICNT *container ;
  char *txt ;
  char *defval ;
  SHORT fldlen ;
  SHORT boxlen ;
  SHORT x, y ;
  SHORT dx, dy ;
  USHORT class ;
{
 /*
 **  Allocheremo sei aree dati:
 **
 **  -- una struttura Gadget
 **  -- una struttura StringInfo
 **  -- una struttura IntuiText
 **  -- una struttura Border
 **  -- un vettore da cinque coordinate
 **  -- un buffer
 **
 **  più l'area di servizio
 */
  IGDG *Gdg ;
  STRI *Str ;
  ITXT *Txt ;
  IBRD *Brd ;
  UFLD *User ;
  SHORT *Coord ;
  UBYTE *Buf ;
  USHORT GdgSize, StrSize, TxtSize, BrdSize, CrdSize,
         UsrSize, BufSize, TotSize ;
  SHORT l, h;

 /*
 **  Filtro di sicurezza
 */
  if (fldlen <= 0 || boxlen <= 0 || fldlen < boxlen) return(NULL) ;

 /*
 **  Calcola le dimensioni delle varie aree
 */
  GdgSize = sizeof(IGDG) ;
  StrSize = sizeof(STRI) ;
  TxtSize = sizeof(ITXT) ;
  BrdSize = sizeof(IBRD) ;
  CrdSize = sizeof(SHORT)*10;
  UsrSize = sizeof(UFLD) ;


 /*
 **  Attenzione: questa normalizzazione è importante. Essa serve a
 **  far si che il buffer sia un multiplo intero di parole (LONG),
 **  cosi da allineare l'area di servizio (UserData) alla voce.
 */
  BufSize = ((fldlen + 1)/4 + 1) * 4;

  TotSize = GdgSize + StrSize + TxtSize + BrdSize + CrdSize +
            UsrSize + BufSize;


 /*
 **  Alloca memoria per il campo e le strutture collegate
 */
  Gdg = (IGDG *)AllocMem(TotSize, GDGMEM) ;
  if (Gdg == NULL) return(NULL) ;
  Str       = POINTER( STRI, Gdg,    GdgSize) ;
  Txt       = POINTER( ITXT, Str,    StrSize) ;
  Brd       = POINTER( IBRD, Txt,    TxtSize) ;
  Coord     = POINTER(SHORT, Brd,    BrdSize) ;
  Buf       = POINTER(UBYTE, Coord,  CrdSize) ;
  User      = POINTER( UFLD, Buf,    BufSize) ;


 /*
 **  Assegna i campi fissi della struttura Gadget
 */
  Gdg->NextGadget = NULL ;          /* Questo è un controllo singolo      */
  Gdg->GadgetID   = id ;            /* Identificativo del controllo       */
  Gdg->GadgetType = STRGADGET ;     /* Tipo di controllo                  */
  Gdg->Flags      = GADGHCOMP ;     /* Unica forma valida per i campi     */
  Gdg->Activation = RELVERIFY ;     /* Mi interessa sapere se rilasciato  */
  if (class == NUMERICFIELDCLASS)   /* Se è un campo numerico, allora     */
    Gdg->Activation |= ( LONGINT |  /*  -  campo di tipo numerico         */
                        STRINGRIGHT ) ; /*  -  allineato a destra          */
  Gdg->MutualExclude = NULL ;       /* Non utilizzato -- per sicurezza -- */
  Gdg->GadgetText   = Txt ;         /* Testo associato al campo           */
  Gdg->GadgetRender = (APTR)Brd ;   /* Cornice del campo                  */
  Gdg->SelectRender = NULL ;        /* Non utilizzato per i campi         */
```

scopo del campo, e quindi che tipo di valori ci si aspetta che vengano introdotti. Ad esempio, se state scrivendo un programma di gestione di una base dati, accanto ad ogni campo per l'introduzione dei valori di un record potreste voler mettere il nome del record stesso (per esempio *Indirizzo, Cognome, Prezzo* e simili).

Per comodità, diamo la possibilità di posizionare indipendentemente il campo ed il testo associatovi. Infatti, sebbene in genere il testo viene messo a sinistra del campo, se il campo è abbastanza lungo potreste decidere di posizionarlo sotto al testo. Oppure potreste utilizzare il testo per indicare all'utente i valori ammessi nel campo, ed in questo caso il testo viene di solito messo a destra del campo (vedi figura 1).

Vediamo allora che parametri passare alla **CreateField()**:

id è, come al solito, l'identificativo del controllo, che serve nel codice di gestione degli eventi emessi da Intuition;

container è il contenitore del campo (finestra o quadro);

txt è il testo associato al campo. Esso può essere eventualmente nullo;

defval è il valore di default del campo, quello cioè a cui il campo viene inizializzato all'atto della visualizzazione del contenitore. Anch'esso può essere nullo.

fldlen è la lunghezza vera del campo, cioè la lunghezza *in caratteri* del campo (escluso il byte di terminazione **\0**), indipendentemente da quanto viene effettivamente visualizzato sullo schermo;

boxlen è la lunghezza visibile del campo, cioè la lunghezza *in caratteri* del rettangolo che rappresenta il campo nel contenitore, e che è a tutti gli effetti una finestra sul campo vero e proprio;

x, y è la posizione del campo nel contenitore. Queste coordinate sono quindi relative all'origine del contenitore, e vengono interpretate secondo la tecnica di posizionamento contestuale già utilizzata per i pulsanti nelle scorse puntate;

dx, dy è la posizione del testo rispetto al campo. Queste coordinate sono relative all'origine del campo, secondo una tecnica contestuale analoga alla precedente, e spiegata più avanti;

class è la classe del campo (tipo campo). Attualmente sono possibili due valori: **STRINGFIELDCLASS** per i campi di tipo stringa, e **NUMERICFIELDCLASS** per quelli numerici.

Due parole sul posizionamento del campo e su quello del testo ad esso associato.

Il posizionamento dell'area di selezione è anche in questo caso sempre relativo ad un bordo del contenitore: quello superiore o di sinistra, per valori positivi delle coordinate, quello inferiore o

```
    Gdg->SpecialInfo  = (APTR)Str ;  /* Puntatore alla struttura StringInfo */
    Str->Buffer       = Buf ;        /* Puntatore all'area dati del campo   */
    Str->UndoBuffer   = NULL ;       /* Puntatore all'area di salvataggio   */
    Str->BufferPos    = 0 ;          /* Posizione del cursore in area dati  */
    Str->MaxChars     = fldlen + 1 ; /* Massimo numero di caratteri (+ \0)  */
    Str->DispPos      = 0 ;          /* Posizione del primo carattere       */
    Str->AltKeyMap    = NULL ;       /* Mappa alternativa della tastiera    */

    sprintf(Buf,"%s\0",defval) ;   /* Copia il valore di default nel buffer */

    Gdg->UserData     = (APTR)User ; /* Questa è una struttura di servizio  */
    User->Number      = 1 ;          /* Numero di campi nel gruppo          */
    User->Size        = TotSize ;    /* Qui metto quanta memoria ho preso   */


    /*
    **  Il campo fa parte di un quadro ?
    */
     if (container->r) Gdg->GadgetType |= REQGADGET ;

    /*
    **  Associa al campo il suo testo
    */
     l = container->w->RPort->TxWidth ;  /* Lunghezza di un carattere        */
     h = container->w->RPort->TxHeight ; /* Altezza di un carattere          */


     *Txt = textModel ;                  /* Copia il prototipo del controllo */
     Txt->IText = (UBYTE *)txt ;         /* Copia il testo vero e proprio    */
     Txt->FrontPen = 3 ;    /* Così il testo usa un colore diverso dal campo */

    /*
    **  Calcola le dimensioni del campo
    */
     Gdg->Width    = boxlen * l ;
     Gdg->Height   = h + 1 ;


    /*
    **  Per rendere più semplice la vita al programmatore, se un campo è
    **  negativo, lo consideriamo una coordinata rispetto al bordo del
    **  controllo PIU' VICINO a quello da cui si parte a misurare.
    */
     if (x > 0)                      /* Ascissa rispetto ad...              */
     {
       x += container->w->BorderLeft ;
       Gdg->LeftEdge = x ;                        /* ...il bordo sinistro */
     }
     else
     {
       x -= container->w->BorderRight ;
       Gdg->LeftEdge = x - Gdg->Width ;           /* ...il bordo destro   */
       Gdg->Flags |= GRELRIGHT ;
     }

     if (y > 0)                      /* Ordinata rispetto ad...             */
     {
       y += container->w->BorderTop ;
       Gdg->TopEdge  = y ;                        /* ...il bordo superiore */
     }
     else
     {
       y -= container->w->BorderBottom ;
       Gdg->TopEdge  = y - Gdg->Height ;          /* ...il bordo inferiore */
       Gdg->Flags |= GRELBOTTOM ;
     }

    /*
    **  La posizione del testo rispetto al campo è calcolata come segue:
    **
    **  - se dx è positivo, rappresenta la posizione del testo a partire
    **    dal fondo del campo, nel senso delle ascisse positive
    **  - se dx è negativo, rappresenta la posizione del testo a partire
    **    dall'inizio del campo, nel senso delle ascisse negative
    **
    **  Analogamente per dy.
    */
     Txt->LeftEdge = dx + ((dx > 0) ? Gdg->Width : 0) ;
     Txt->TopEdge = dy + ((dy > 0) ? Gdg->Height : 0) ;

    /*
    **  Definisci la struttura bordo
    */
     *Brd = rectModel ;

     Brd->FrontPen = 2;
     Brd->NextBorder = NULL ;
     Brd->XY = Coord ;

    /*
    **  Spostiamo l'origine del bordo di due pixel in alto a sinistra, per
    **  evitare che la cornice del campo tocchi il testo in esso contenuto.
    */
     Brd->LeftEdge -= 2 ;
     Brd->TopEdge  -= 2 ;

    /*
    **  Definisci i vettori di coordinate
    */
     Coord[2] = Gdg->Width + 2 ;  Coord[4] = Gdg->Width + 2 ;
     Coord[5] = Gdg->Height + 1 ; Coord[7] = Gdg->Height + 1 ;

    /*
    **  Fatto. Il campo è pronto.
    */
     return (Gdg) ;
    }
```

Figura 3 — CreateField().

di destra, per valori negativi delle stesse. Ricordo che si tratta di una convenzione nostra, atipica nell'ambiente Amiga, ma molto più semplice per calcolare le coordinate quando si scrive il codice.

Questa tecnica, detta contestuale, in quanto il tipo di posizionamento dipende dal contesto, e precisamente dal segno delle coordinate, è stata ampiamente spiegata nelle prime puntate dedicate ai pulsanti.

Per quello che riguarda il testo associato al campo, opereremo in modo analogo. Questa volta, tuttavia, la convenzione è leggermente differente. E precisamente: se le coordinate del testo sono positive, il testo è posizionato rispetto al bordo destro ed a quello inferiore dell'area di selezione del campo, come mostrato in figura 2, altrimenti esso è posizionato relativamente ai bordi sinistro e superiore del campo. In pratica, in caso una coordinata del testo è positiva, ad essa viene aggiunta la dimensione del campo nella stessa direzione (larghezza per le ascisse ed altezza per le ordinate).

Facendo ora riferimento alla figura 3, vediamo adesso la logica della funzione.

Il primo blocco è il solito filtro di sicurezza. In questo caso ci siamo limitati a verificare che la lunghezza visibile del campo e quella vera siano entrambe strettamente positive, e che la prima sia inferiore od uguale alla seconda. Quest'ultima verifica può essere eliminata se si ammette di avere campi più corti dell'area di selezione dello stesso. Nel nostro caso si è deciso di mantenerla.

Il secondo blocco serve a calcolare, come di consueto, le dimensioni dell'area di memoria destinata ad ospitare le strutture relative al controllo. Questa volta le strutture sono sei: una per il controllo vero e proprio **(Gdg)**, una per le informazioni speciali relative al campo **(Str)**, una per il testo associato al campo **(Txt)**, una per il bordo che incornicia l'area di selezione **(Brd)**, una per le coordinate del bordo stesso **(Coord)**, ed infine una per la solita area di servizio **(User)**. Quest'ultima, stavolta, conterrà una struttura differente da quella utilizzata per i pulsanti, e cioè la **UsrField** (vedi figura 6).

In questo blocco c'è una istruzione di normalizzazione che si è rivelata necessaria in fase di prova *[test]* della procedura. Mentre infatti la maggior parte delle strutture di Intuition hanno una dimensione pari ad un numero intero di parole da due byte, l'area dati del campo può essere definita come un numero dispari di byte. In questo caso la struttura di servizio **User** risulta non allineata alla parola corta **(short)**, e questo manda in *GURU* il programma, quando si prova ad assegnare un valore ad uno qualunque dei suoi campi. Per evitare

```
/*************************************************************************
** DeleteField()                FUNZIONE            Versione 1.00       **
**                                                                       **
** Funzione fornita:  cancella dinamicamente un campo di I/O             **
**                                                                       **
** Dati in ingresso:  field      puntatore al campo                      **
**                    container  contenitore (finestra e/o quadro)       **
**                                                                       **
** Dati in uscita:    nessuno                                            **
**                                                                       **
** Dati globali:      User       ricava da Size la memoria utilizzata    **
**                               e da Number il numero di campi          **
**                                                                       **
*************************************************************************/

void DeleteField(field,container)
  IGDG *field ;
  ICNT *container ;
{
 /*
 **  Rimuovi il campo dal contenitore
 */
  (void)RemoveGList(container->w,field,
                    ((UFLD *)(field->UserData))->Number);
  RefreshWindow(container->w) ;

 /*
 **  Cancella la memoria per il campo e le strutture collegate
 */
  FreeMem(field,(((UFLD *)(field->UserData))->Size)) ;
}
```

Figura 4 — DeleteField().

```
/*************************************************************************
** DisplayField()               FUNZIONE            Versione 1.00       **
**                                                                       **
** Funzione fornita:  aggiunge dei campi al contenitore e quindi li      **
**                    visualizza                                         **
**                                                                       **
** Dati in ingresso:  field      puntatore ai campi    (uno o gruppo)    **
**                    container  contenitore (finestra e/o quadro)       **
**                                                                       **
** Dati globali:      User       ricava da Number il numero di campi     **
**                                                                       **
*************************************************************************/
void DisplayFields(field,container)
  IGDG *field ;
  ICNT *container ;
{
  USHORT n ;

 /*
 **  Aggiungi il campo al contenitore
 */
  n = ((UFLD *)(field->UserData))->Number ;
  (void)AddGList(container->w,field,EOGL,n,container->r) ;
  RefreshGList(field,container->w,container->r,n) ;
}
```

Figura 5 — DisplayField ().

ciò, ho deciso di allineare la **UsrField** addirittura ai quattro byte (o parola lunga), come appunto mostrato nel listato. L'area dati risulta così più lunga, ma ad Intuition viene fornita comunque la lunghezza richiesta del campo. Esiste tuttavia un effetto collaterale che potrebbe essere dovuto a questa inconsistenza tra la lunghezza reale dell'area dati, e quella dichiarata ad Intuition. Se si riempie il campo fino in fondo, nel caso di campi più lunghi dell'area visualizzata, e poi si fa scorrere indietro il cursore per poi riportarlo a fondo campo, compare un carattere fantasma in fondo al campo *non editabile*. Esso in realtà non viene riportato nell'area dati, ma appare solo sullo schermo. Se volete, provate a capire perché questo succede, ed eventualmente eliminate la normalizzazione spostando però **Buf** dopo **User**.

Il terzo blocco alloca la memoria calcolata e posiziona le strutture in essa, utilizzando la macro **POINTER()**.

Il quarto blocco di codice definisce il controllo come un campo, eventualmente numerico, nel caso che **class** abbia il valore **NUMERICFIELDCLASS**, ed assegna la maggior parte dei campi di **Gdg** ed **Str**. Da notare che si è deciso di allineare a sinistra il contenuto dei campi stringa, a destra quello dei campi numerici.

Verso la fine di questo quarto blocco, c'è una istruzione che carica il valore di *default* del campo nell'area dati, aggiungendovi un ulteriore byte nullo, come richiesto da Intuition (vedi la scorsa puntata).

I blocchi successivi verificano se il campo fa parte di un quadro, associano al campo il suo testo e calcolano le dimensioni dell'area visibile del campo.

Ricordo che entrambe le lunghezze del campo, visibile e vera, erano state fornite in numero di caratteri, quindi vanno convertite in pixel utilizzando le informazioni della struttura **RastPort** del contenitore.

A questo punto c'è il solito calcolo per il posizionamento contestuale del controllo, e quello analogo per il suo testo. Viene infine definita la struttura bordo (una sola, dato che l'unica tecnica di evidenziazione valida è quella *a colore complementare*), ed il gioco è fatto.

Un'ultima considerazione. Supponiamo di non voler associare alcun testo al campo. Perché allora allocare una struttura **ITXT** e sprecare così spazio prezioso? Semplice. Per avere l'opportunità in futuro di scrivere eventualmente una procedura capace di modificare dinamicamente il testo associato ad un campo. Chissà. Potrebbe risultare utile...

DeleteField()

La funzione di rimozione di un campo (figura 4) è analoga a quella usata per i pulsanti. L'abbiamo riscritta piuttosto che ridefinire la precedente via macro, sia perché essa fa riferimento alla nuova struttura **UsrField**, sia perché in futuro essa potrebbe essere chiamata a gestire alcune caratteristiche di un campo che non hanno un equivalente per i pulsanti. Questo è lo stesso motivo per il quale non abbiamo usato la struttura di servizio dei pulsanti anche per i campi.

DisplayFields()

Un discorso analogo può essere fatto per la **Display Fields()** (figura 5), strutturata per il momento come la vecchia **DisplayButtons()** e sulla quale non spenderemo quindi altro tempo.

Esercizio

Prima di concludere, vi lascio con un semplice esercizio, di cui però non vi darò la soluzione nella prossima puntata, dato che esistono infiniti modi di svolgerlo, tutti altrettanto validi.

Uno dei principali problemi con i campi, è quello di verificare i dati in ingresso con un insieme di dati validi per quel campo. Ad esempio, sebbene in un campo numerico Intuition mi permette di introdurre valori compresi fra *−2.147.483.648* e *2.147.483.647*, in genere il valore da introdurre appartiene ad un insieme più ristretto, come ad esempio nel caso delle date, o detta temperatura ambiente. Lo stesso vale per i campi stringa, anzi, in quel caso il tipo di controlli da effettuare sul campo può essere alquanto complesso. Senza contare che in certi casi, i valori ammessi per un campo dipendono da quelli immessi in un altro campo. Ad esempio, il giorno in una data, fa parte di un insieme differente a seconda del mese immesso e del fatto se l'anno sia bisestile o meno.

Come si intuisce facilmente, è impossibile creare delle funzioni di controllo valide per tutti i casi possibili. Può essere comodo tuttavia costruirsi un insie-

```
/*
**  Definizioni da precompilare per generare la tabella GDGUSRH.SYM
*/

/*
**  Costanti
*/
 #define TOGGLEON  1
 #define TOGGLEOFF 0
 #define AUTOBUTTONCLASS     0x0001
 #define TOGGLEBUTTONCLASS   0x0002
 #define ROWBUTTONGROUP      0x0003
 #define COLBUTTONGROUP      0x0004
 #define ALPHAFIELDCLASS     0x0001
 #define NUMERICFIELDCLASS   0x0002


/*
**  Tipi
*/
 typedef struct Gadget       IGDG;
 typedef struct Border       IBRD;
 typedef struct StringInfo   STRI;

 typedef struct Container
 {
   struct Window *w;
   struct Requester *r;
 }
 ICNT;

 typedef struct UsrButton
 {
   USHORT Number ;
   USHORT Size ;
   APTR First, Selected ;
   ULONG Total, Counter ;
 }
 UBUT;

 typedef struct UsrField
 {
   USHORT Number ;
   USHORT Size ;
 }
 UFLD;

/*
**  Macro di servizio
*/
 #define CreateAutoButton(id,txt,cont,x,y) \
    CreateButton((id),(txt),(cont),(x),(y),AUTOBUTTONCLASS,TOGGLEOFF)
 #define DeleteAutoButton(b,c) DeleteButton((b),(c))
 #define CreateToggleButton(id,txt,cont,x,y,status) \
    CreateButton((id),(txt),(cont),(x),(y),TOGGLEBUTTONCLASS,(status))
 #define DeleteToggleButton(b,c) DeleteButton((b),(c))
 #define DeleteXButtons(b,c) DeleteButton((b),(c))
 #define DeleteSpinButton(b,c) DeleteButton((b),(c))
 #define ToggleButtonStatus(b)    (BOOL)((b)->Flags & SELECTED)
 #define NumericFieldValue(f) (((STRI *)((f)->SpecialInfo))->LongInt)
 #define StringFieldValue(f) (((STRI *)((f)->SpecialInfo))->Buffer)


/*
**  Prototipi delle funzioni di servizio
*/
 IGDG *CreateButton    ( USHORT, char *, ICNT *, SHORT, SHORT, USHORT, BOOL );
 void  DeleteButton    ( IGDG *, ICNT * );
 IGDG *CreateXButtons  ( USHORT, USHORT, char **, ICNT *, SHORT, SHORT, \
                         USHORT, USHORT );
 IGDG *SelectXButtons  ( IGDG *, ICNT * );
 IGDG *CreateSpinButton( USHORT, USHORT, char **, ICNT *, SHORT, SHORT, \
                         USHORT, USHORT );
 char *SelectSpinButton( IGDG *, ICNT * );
 void  DisplayButtons  ( IGDG *, ICNT * );
 void  RefreshWindow   ( struct Window *);
 IGDG *CreateField     ( USHORT, ICNT *, char *, char *, \
                         SHORT, SHORT, SHORT, SHORT, SHORT, SHORT, USHORT );
 void  DeleteField     ( IGDG *, ICNT * );
 void  DisplayFields   ( IGDG *, ICNT * );
```

Figura 6 — gdgusrh.c.

```
BOOL isBetween( LONG valore, LONG inferiore, LONG superiore) ;

Verifica se inferiore <= valore <= superiore
Se si torna TRUE, altrimenti FALSE

Esempio:

isBetween(234,-12,678) è TRUE

isBetween(0,12,24) è FALSE

NOTA: può anche essere implementata come una macro. In questo caso si
consiglia di usare il cast (LONG) davanti a tutti i valori interi.
```

```
BOOL isMadeof( char *valore, char *caratteri) ;

Verifica se per la stringa "valore" sono stati usati solo i caratteri
riportati nella stringa "caratteri".
Se si torna TRUE, altrimenti FALSE

Esempio:

allMixed = "aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ"

isMadeof("Dario de Judicibus",allMixed) è TRUE

isMadeof("0.45E14","1234567890.+-") è FALSE

NOTA: si può pensare di usare anche una sintassi più complessa per la
stringa di validazione, come ad esempio quella usata da GREP.
In questo caso allMixed avrebbe potuto essere scritto più semplice-
mente come segue: [a-zA-Z]
```

```
BOOL isAbbrev ( char *valore, char *parola, char *minimo ) ;

Verifica se per la stringa "valore" è una abbreviazione valida della
stringa "parola", dove "minimo" rappresenta il minimo numero di
caratteri affinchè l'abbreviazione sia considerata valida.
Se si torna TRUE, altrimenti FALSE

Esempio:

isAbbrev("OPTS","OPTIONS",3) è FALSE

isAbbrev("Q","QUIET",1) è TRUE

NOTA: si può aggiungere anche un quarto parametro ad indicare se il
fatto se i caratteri siano maiuscoli o minuscoli va ignorato oppure
rispettato, come segue:

BOOL isAbbrev ( char *valore, char *parola, char *minimo, BOOL case ) ;
```

```
BOOL isAmong ( char *valore, char *parola[] ) ;

Verifica se per la stringa "valore" è una parola fra quelle riportate
nel vettore di stringhe "parola".
Se si torna TRUE, altrimenti FALSE

Esempio:

options[] =
{
    "QUIET"
  , "CLONE"
  , "ASCII"
  , "BINARY"
} ;

isAmong("ASCII",options) è TRUE

isAmong("ALL",options) è FALSE

NOTA: si può aggiungere anche un terzo parametro ad indicare se il
fatto se i caratteri siano maiuscoli o minuscoli va ignorato oppure
rispettato, come segue:

BOOL isAmong ( char *valore, char *parola[], BOOL case ) ;
```

Figura 7 — Funzioni di validazione.

me di funzioni atomiche per verificare i casi più comuni. In figura 7 vi propongo i prototipi di alcune di queste funzioni. A voi il compito di implementarle come meglio credete, cercando possibilmente di sviluppare un codice efficiente e veloce. Chi conosce l'*Assembler* può anche pensare di scriverle in questo linguaggio, se crede, dato che è molto adatto a scrivere algoritmi per la comparazione di dati. Naturalmente potete continuare sulla stessa falsariga e creare molte altre funzioni che vi potranno tornare utili in seguito. Chi conosce l'*AREXX* può inoltre pensare di sfruttare le potenti funzioni per la manipolazione delle stringhe fornite dalle librerie di questo linguaggio per semplificarsi la vita. Viceversa, potreste provare a scrivere voi una vostra libreria dinamica di funzioni di validazione, da distribuire come *Public Domain*.

Conclusione

Fine della seconda puntata dedicata ai campi. Nella prossima vedremo come si usano le procedure viste, e come si ricava dal campo il valore immesso dall'utente. Buon lavoro! MC

Casella Postale

Vi ricordate di Davide Ficano e della sua procedura per assicurarsi di partire sempre con uno schermo PAL su macchine di questo tipo? Era il lontano 1990, e la puntata era la 27ª. Si trattava di una piccola routine Assembler che, nelle intenzioni dell'autore, doveva risolvere il problema per gli A500 e gli A2000 (per l'A1000 il problema era già stato affrontato nella *Scheda Tecnica della 16ª puntata).* Beh, qualche puntata più tardi, la 31ª per la precisione, Nicola Salmoria affettava a tocchetti il povero Davide, a causa di una non precisa pulizia del programma in questione.

Ce n'era abbastanza per scoraggiare anche il programmatore più incallito, ma Davide Ficano ha dimostrato di apprezzare la critica di Nicola Salmoria per quello che era: una segnalazione intesa a migliorare la routine in questione, ed a salvaguardare quella disciplina nel programmare che fa di un programma *un buon programma.*

Cosa dire? Solo che c'è da prendere esempio da entrambi. E speriamo che stavolta tutto giri perfettamente...

```
; 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 A — Controllore di schermo ☐ versione BOOT.

Figura B — Controllore di schermo ☐ versione CLI.

```
; 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
```

Procedura Assembler per il controllo dello schermo

Egregio Dott. de Judicibus,
ho letto la lettera del Sig. Salmoria e la sua risposta, e non posso che essere d'accordo con tutti e due. Chiedo scusa a tutti gli utenti Amiga, a lei Dott. de Judicibus, ma soprattutto al Sig. Salmoria che a quanto pare ha provato la routine con spiacevoli risultati. Io non mi sono ASSOLUTAMENTE risentito, anzi per me è stato uno spunto per migliorare la routine e per confrontare il mio «stile» (o forse è meglio chiamarlo «non stile») di programmazione con quello di altri.

Le critiche, quando sono costruttive non possono che far bene al carattere, e questa mi ha fatto proprio bene (SIGH!!). Forse con la lettera del Sig. Salmoria la mia routine ha assunto un valore didattico superiore perché mostra come NON SI DEVE PROCEDERE (povero illuso direte voi!).

Comunque dato che il danno l'ho fatto io è giusto che io ponga rimedio e presento la routine che dovrebbe (adesso il condizionale è d'obbligo) funzionare su tutte le macchine, come al solito presento una versione CLI ed una da mettere sul BOOT.

La routine l'ho provata su un 2000 con 3 Mega ed ha fatto il suo dovere al contrario della precedente. Se dovessero esserci ancora problemi spero che li affronteremo insieme da buona comunità Amiga.

Per i linking, le compilazioni e altro valgono le stesse identiche regole presentate nel nr. 101. (NdA. La versione BOOT è riportata in figura A, mentre la versione CLI è riportata in figura B).

Distinti saluti

Davide Ficano — Palermo

MC

HABER CO., LTD

TAIPEI, TAIWAN, R.O.C.

MOTHERBOARDS

Legal BIOS, OK, DIP/SIMM Ram, test & burn-in		
HP-3012	*286-12, Headland, exp. 4M, half*	*113.900*
HP-3013	*286-16, Headland, exp. 4M, half*	*137.300*
HP-3003	*386SX-16, C&T, exp. 8M, baby*	*482.900*
HP-3004	*386-25, C&T, exp. 8M, baby*	*756.800*
HP-3017	*386-33, SIS, exp. 16M, cache, baby*	*967.000*
HP-3008	*486-33, ISA, Intel, exp. 64k, cache, baby*	*2.290.000*
HP-3010	*486-33, EISA, Intel, exp. 64M, cache, full*	*3.963.000*

UPS

HP-1000	*400VA, mini, TÜV approved*	*354.000*
HP-1001	*600VA, mini, TÜV approved*	*390.000*
HP-1002	*1500VA, mini, TÜV approved*	*830.000*
	Novell monitor, card & sftwr	*130.000*

CARDS

HP-4002	*IDE, ATbus + 2s/p/g, cables*	*30.600*
HP-4012	*VGA, OAK, 512k, 1024 + portrait, I/NI*	*105.200*
HP-4015	*VGA, Tseng, 512k, 1024*	*135.000*
HP-4014	*VGA, Trident, 1M, 1024/256 col.*	*171.000*
HP-4008	*Ethernet, 16bit, NE2000 comp.*	*185.000*
	NETWARE ver. 2.2	*call*

CASES, KEYBOARDS

HP-6003	*desktop w/power*	*103.000*
HP-6006	*desktop slim w/power*	*138.000*
HP-6004	*mini tower w/power*	*158.000*
HP-6002	*super tower*	*267.400*
HP-9003	*102k keyboard*	*40.500*

COMPONENTS

FD1.2/1.44, HD ATbus 40-200MB, mouses bus & serial, LAN cards, UNIX/XENIX intelligent 8/16 serial cards, industrial I/O cards.

Assembled and tested systems in all the configuration

Stock e Garanzia in Italia

RISERVATO AD OPERATORI DEL SETTORE

HABER Co., Ltd (Italia) - Tel. 0321/399457 - Fax 0321/35061