Programmare in C su Amiga (34)

di Dario de Judicibus
(MC2120)

Introduzione

Ultima puntata dedicata ai pulsanti. Terminiamo questa serie di puntate con un nuovo tipo di controllo: i pulsanti a rotazione.

Nella scorsa puntata abbiamo visto come sia possibile usare la struttura **Gadget** di Intuition per creare un tipo di controllo non standard, non previsto cioè dal sistema operativo, ma perfettamente legale: *il gruppo di pulsanti a rilascio incrociato*. Per far questo abbiamo sfruttato la possibilità che ci dà Intuition di estendere la struttura **Gadget** con una struttura di servizio che abbiamo chiamato **UsrButton.** In effetti avevamo utilizzato questa tecnica anche con i due tipi «classici» di pulsanti, e cioè quelli *a rilascio automatico* e quelli *a rilascio manuale*. Tuttavia, mentre per i due tipi standard la struttura **UsrButton** era stata utilizzata solo per mantenere in essa informazioni di comodo, nel caso del gruppo di pulsanti mutualmente esclusivi tale struttura è risultata fondamentale ai fini di un corretto funzionamento dell'intero gruppo come un *singolo* controllo. Questo in quanto Intuition continua a vedere i singoli pulsanti come entità separate, per cui tocca al programma gestire direttamente il controllo, utilizzando appunto le informazioni contenute nella struttura di servizio.

Concludiamo ora questa serie di articoli dedicati ai pulsanti, mostrando come si possa creare un altro tipo di pulsante, peraltro presente nella versione 2.0 del sistema operativo (ma non nella 1.3, che resta ancora la più diffusa nel mondo Amiga).

Inoltre risponderemo al *quiz* presentato lo scorso mese, e vi proporremo un esercizio finale per valutare il vostro livello di acquisizione della tecnica appresa in questi ultimi mesi.

Gruppi di pulsanti

Abbiamo visto che i pulsanti a rilascio incrociato ci permettono di scegliere una possibilità fra molte, tutte mutualmente esclusive. Ad esempio il colore del fondo dello schermo, fra quelli disponibili nei registri associati allo schermo. Oppure la modalità di trasferimento di un file tra due sistemi differenti e parzialmente compatibili, come l'Amiga DOS e l'MS-DOS, e cioè trasferimento *binario* o di tipo *testo* (ASCII). E via dicendo.

Questo tipo di controllo ha tuttavia uno svantaggio: *occupa troppo spazio*. Supponiamo infatti di dover proporre all'utente tutta una serie di opzioni relative al trasferimento di file via modem, possibilmente nello stesso quadro. Una specie di visione di insieme. Ovviamente in questo caso, il numero di gruppi di pulsanti a rilascio incrociato è alquanto elevato. Abbiamo le varie velocità di trasferimento, da 300 a 9600 baud, i protocolli di trasferimento (XModem, YModem, ZModem, e molti altri), la parità, il numero di bit, e così via. Inutile dire che metterli tutti nello stesso pannello è un'impresa (vedi figura 1). Bisogna inoltre tenere presente che, oltre ai pulsanti stessi, sarebbe opportuno mettere nel quadro un titolo per ogni gruppo, se non addirittura una cornice che delimiti il gruppo stesso. Quest'ultima potrebbe essere addirittura inclusa nella **CreateXButtons()** come elemento stilistico standard.

A questo punto ci sono tre alternative:

1. suddividere il quadro delle opzioni in più quadri, ognuno richiamabile da un pannello principale;
2. utilizzare dei radio-pulsanti [*radio buttons*] come fa *Windows 3.0* od il *Presentation Manager* dell'OS/2.

Figura 1
Affollamento eccessivo di pulsanti.

Figura 2
Radio-pulsanti.

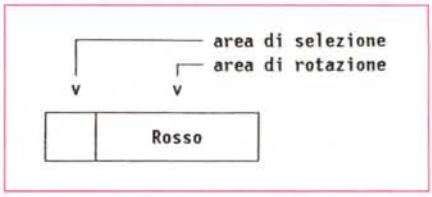


Figura 3
Pulsante a rotazione.

```
/*
**  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
/*
**  Tipi
*/
 typedef struct Gadget       IGDG;
 typedef struct Border       IBRD;

 typedef struct Container
 {
   struct Window *w;
   struct Requester *r;
 }
 ICNT;
typedef struct UsrButton  /* Cambiamenti nella struttura UsrButton vvvvvvv */
{
  USHORT Number ;
  USHORT Size ;
  APTR First, Selected ;  /* I campi IGDG* sono stati convertiti ad APTR   */
  ULONG Total, Counter ;  /* Due nuovi campi per i pulsanti a rotazione    */
}
UBUT;
```

```
/*
**  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))
 /* ----- NUOVA MACRO ------------------------------------------------- */
 #define DeleteSpinButton(b,c) DeleteButton((b),(c))
 /* ------------------------------------------------------------------- */
 #define ToggleButtonStatus(b)   (BOOL)((b)->Flags & SELECTED)

/*
**  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 * );
 /* ----- NUOVE FUNZIONI ---------------------------------------------- */
 IGDG *CreateSpinButton( USHORT, USHORT, char **, ICNT *, SHORT, SHORT, USHORT, USHORT );
 char *SelectSpinButton( IGDG *, ICNT * );
 /* ------------------------------------------------------------------- */
 void  DisplayButtons( IGDG *, ICNT * );
 void  RefreshWindow ( struct Window *);
```

Figura 4 - gdgusrh.c.

3. utilizzare dei *pulsanti a rotazione* [*spin buttons*], come nella versione 2.0 del sistema operativo.

I radio-pulsanti, non sono altro che dei pulsanti mutualmente esclusivi come quelli che abbiamo visto nelle ultime due puntate. Cambia solo lo stile del pulsante (vedi figura 2). Il testo infatti non è incorniciato da un bordo rettangolare, ma è affiancato da un piccolo pulsante circolare grande più o meno come un carattere maiuscolo. Quando il pulsante è selezionato, l'interno del cerchietto diventa nero. Ovviamente, se il pulsante non è selezionato, il cerchietto è vuoto.

Questo stile ha due vantaggi rispetto a quello più classico a pulsante:

- una maggiore possibilità di compattazione dei vari elementi, dovuta alla mancanza del bordo intorno al testo;
- una maggiore flessibilità nel testo da associare al pulsante, che può adesso essere rappresentato anche da una frase o comunque una stringa di più parole, piuttosto che da una o due parole, come è uso fare nel caso di pulsanti dotati di bordo rettangolare, per evidenti ragioni pratiche.

Al di là di queste differenze stilistiche, tuttavia, un gruppo di radio-pulsanti (detti anche *bottoni* a causa della loro forma circolare), non si comporta in modo sostanzialmente differente dai pulsanti a rilascio incrociato che abbiamo visto in precedenza. L'unica differenza è l'utilizzo di una maschera circolare da applicare all'area di selezione del pulsante, in modo che questa combaci con l'immagine del cerchietto utilizzata per rendere visibile tale area. Per questo motivo non ho ritenuto necessario trattare lo sviluppo del codice relativo ai radio-bottoni, dato che esso si può facilmente ricavare modificando la **CreateXButtons()**, ottenendo così la **CreateRadioButtons().** Addirittura le due funzioni di cancellazione e selezione **DeleteRadioButtons()** e **SelectRadioButtons()** sono esattamente le stesse delle loro equivalenti per i pulsanti a rilascio incrociato, per cui basta ridefinirle con un paio di istruzioni #**define.**

Questa è proprio l'esercizio menzionato nell'introduzione, e che sicuramente risolverete in pochissimo tempo, se avete seguito con attenzione gli ultimi quattro articoli di questa serie.

Vediamo invece che cosa è un pulsante a rotazione. Come al solito, costruiremo tre funzioni, una di creazione (**CreateSpinButton()**), una di cancellazione (**DeleteSpinButton()**), ed una di selezione (**SelectSpinButton()**).

Pulsanti a rotazione

Un pulsante a rotazione [*spin button*] è un singolo pulsante diviso in due parti (vedi figura 3); l'area di *selezione*, e l'area di *rotazione*.

Ad ogni pulsante è associato un insieme *ordinato* di testi, corrispondenti a quelli che in un gruppo di pulsanti a rilascio incrociato sono associati ad ogni singolo pulsante del gruppo.

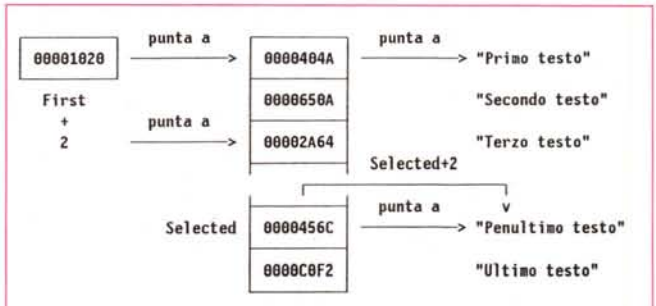
Il testo selezionato è visualizzato nell'area di rotazione, la quale non è mai vuota. Quando l'utente fa click nell'area di selezione, il testo successivo viene visualizzato, e così via a rotazione, fino a tornare al testo iniziale.

In alcuni tipi di pulsanti a rotazione l'area di selezione è divisa in due parti, una che fa scorrere il vettore dei testi in avanti, ed una che fa scorrerlo nel senso opposto. In altri tipi, lo scorrimento avviene sempre nello stesso senso. Ovviamente in questo caso il vettore è sempre ciclico, cioè l'ultimo elemento è agganciato al primo, cosa comune anche per i pulsanti bidirezionali, ma non strettamente necessaria.

In questo articolo ho deciso di mostrare una possibile implementazione di un pulsante a rotazione di tipo *ciclico e monodirezionale*.

Prima di entrare in dettaglio nel codice, un'ultima considerazione. Se da una

Figura 5 - Puntatori.



```
/*************************************************************************
** CreateSpinButton()            FUNZIONE              Versione 1.00     **
**                                                                       **
** Funzione fornita:  crea dinamicamente un pulsante a rotazione         **
**                                                                       **
** Dati in ingresso:  id          identificativo del pulsante            **
**                    n           numero di pulsanti                     **
**                    txt         vettore di testi                       **
**                    container   contenitore (finestra e/o quadro)      **
**                    x, y        posizione del pulsante nel contenitore **
**                    class       classe del pulsante (NULL) [riservato] **
**                    txton       testo inizialmente selezionato         **
**                                                                       **
** Dati in uscita:    Gdg         puntatore al pulsante                  **
**                                                                       **
** Dati globali:      User        memorizza in Size la memoria utilizzata **
**                                in Number uno, in Total il numero      **
**                                totale dei testi associati al pulsante **
**                                ed in Counter l'indice del testo       **
**                                iniziale                               **
**                                                                       **
** Eventi emessi:     GADGETUP                                           **
**                                                                       **
*************************************************************************/
IGDG *CreateSpinButton(id,n,txt,container,x,y,class,txton)
  USHORT id ;
  USHORT n ;
  char *txt[] ;
  ICNT *container ;
  SHORT x, y ;
  USHORT class ;
  USHORT txton ;
{
 /*
 **  Allocheremo quattro aree dati:
 **
 **  -- una struttura Gadget
 **  -- una struttura IntuiText
 **  -- una struttura Border
 **  -- un vettore di sette coordinate
 **
 */
  IGDG *Gdg ;
  ITXT *Txt ;
  IBRD *SpinBrd ;
  UBUT *User ;
  SHORT *SpinCoord ;
  USHORT GdgSize, TxtSize, BrdSize, CrdSize, UsrSize, TotSize ;
  SHORT i, l, h ;
  USHORT n1 ;

 /*
 **  Filtro di sicurezza
 */
 n1 = n - 1;
 if (n < 2 || txton < 0 || txton > n1) return(NULL) ;

 /*
 **  Calcola le dimensioni delle varie aree
 */
 GdgSize = sizeof(IGDG) ;
 TxtSize = sizeof(ITXT) ;
 BrdSize = sizeof(IBRD) ;
 CrdSize = sizeof(SHORT)*14 ;
 UsrSize = sizeof(UBUT) ;
 TotSize = GdgSize + TxtSize + BrdSize + CrdSize + UsrSize ;

 /*
 **  Alloca memoria per il pulsante e le strutture collegate
 */
 Gdg = (IGDG *)AllocMem(TotSize, GDGMEM) ;
 if (Gdg == NULL) return(NULL) ;
 Txt       = POINTER( ITXT, Gdg,       GdgSize) ;
 SpinBrd   = POINTER( IBRD, Txt,       TxtSize) ;
 SpinCoord = POINTER(SHORT, SpinBrd,   BrdSize) ;
 User      = POINTER( UBUT, SpinCoord, CrdSize) ;

 /*
 **  Assegna i campi fissi della struttura Gadget
 */
 Gdg->NextGadget = NULL ;          /* Questo è un controllo singolo      */
 Gdg->GadgetID   = id ;            /* Identificativo del controllo       */
 Gdg->GadgetType = BOOLGADGET ;    /* Tipo di controllo                  */
 Gdg->Flags      = GADGHCOMP ;     /* Bordo evidenziato con complemento  */
 Gdg->Activation = RELVERIFY ;     /* Rilascio automatico con rotazione  */
 Gdg->MutualExclude = NULL ;       /* Non utilizzato -- per sicurezza -- */
 Gdg->SpecialInfo   = NULL ;       /* Non utilizzato -- per sicurezza -- */
 Gdg->GadgetText   = Txt ;
 Gdg->GadgetRender = (APTR)SpinBrd ;
 Gdg->SelectRender = (APTR)NULL ;
 Gdg->UserData     = (APTR)User ;  /* Questa è una struttura di servizio */
 User->Size        = TotSize ;     /* Qui metto quanta memoria ho preso  */
 User->Number      = 1 ;           /* Numero di pulsanti nel gruppo      */
 User->First       = (APTR)txt;      /* Punta al vettore dei testi       */
 User->Total       = (ULONG)n ;      /* Numero dei testi                 */
 User->Counter     = (ULONG)txton ;  /* Indice del testo selezionato     */

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

 /*
 **  Trova la lunghezza massima dei vari testi
 */
 *Txt = textModel ;                /* Copia il prototipo del controllo */
 for (i = 0, l = 0; i < n; i++)
 {
   Txt->IText = (UBYTE *)txt[i] ;  /* Copia il testo vero e proprio    */
   l = max(l,ITXTL(Txt)) ;         /* Lunghezza del testo              */
 }

 /*
 **  Quale testo è selezionato in partenza ?
 */
 Txt->IText = (UBYTE *)txt[txton] ;  /* Copia il testo vero e proprio  */

 h = container->w->RPort->TxHeight ; /* Altezza del testo              */

 /*
 **  L'area di selezione è temporaneamente calcolata in modo da coprire
 **  anche il testo, in modo da favorire i calcoli per il posizionamento
 **  relativo più sotto. Poi sarà ridotta ad un quadrato.
 */
 Gdg->Height   = 5*h/3 ;
 Gdg->Width    = 5*l/4 + 2*Gdg->Height;

 /*
 **  Tutti i testi sono allineati a sinistra (Nota l'uso di Height invece
 **  di Width, la quale ha un valore temporaneo)
 */
 Txt->LeftEdge = l/8 + 2*Gdg->Height ;/* Ascissa del testo nel controllo */
 Txt->TopEdge  = h/3 ;               /* Ordinata del testo nel controllo */

 /*
 **  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 ;
 }

 /*
 **  Definisci la strutture bordo
 */
 *SpinBrd = rectModel ;

 SpinBrd->Count = 7 ;              /* Rettangolo + quadrato            */
 SpinBrd->FrontPen = 1 ;
 SpinBrd->NextBorder = NULL ;
 SpinBrd->XY = SpinCoord ;

 /*
 **  Definisci i vettori di coordinate
 */
 SpinCoord[2]  = Gdg->Width - 1 ;
 SpinCoord[4]  = SpinCoord[2] ;
 SpinCoord[5]  = Gdg->Height - 1 ;
 SpinCoord[7]  = SpinCoord[5] ;
 SpinCoord[13] = SpinCoord[5] ;
 SpinCoord[10] = 2*Gdg->Height - 1 ;
 SpinCoord[12] = SpinCoord[10] ;

 /*
 **  ORA possiamo rendere quadrata l'area di selezione
 */
 Gdg->Width  = 2*Gdg->Height ;

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

Figura 6 - CreateSpinButton().