Programmare in C su Amiga (23)

di Dario de Judicibus
(MC2120)

La possibilità di definire due o più voci come mutualmente esclusive ci permette di dare ai nostri menu una maggiore capacità di controllo sulle operazioni effettuate dall'utente. Vedremo inoltre come risolvere il problema della gestione dei testi alternati senza impattare le caratteristiche delle funzioni di servizio

Introduzione

Nella scorsa puntata abbiamo dato allo scheletro di programma su cui stiamo lavorando da qualche mese una struttura più flessibile. Questa struttura ci sarà molto utile quando, nella prossima puntata, introdurremo altri due file per la gestione dei messaggi di errore. Per il momento chiudiamo il discorso relativo alla struttura **MenuItem** parlando del campo **MutualExclude**, come promesso in precedenza. vedremo inoltre come risolvere il problema evidenziato nella 22ª puntata relativo all'allocazione di memoria di strutture **IntuiText** per i testi alternati di alcune voci, inclusa impropriamente nella funzione di servizio **SetupItemList()**. Tale scelta, temporaneamente adottata per introdurre la tecnica di evidenziazione di una voce via **SelectFill**, aveva in realtà lo scopo di mostrare quanto sia facile, anche in un ottica di strutturazione del programma in moduli funzionali, limitare i vantaggi che tale tecnica di programmazione comporta utilizzando scorciatoie che, benché sul momento rappresentino la soluzione più semplice, alla lunga finiscono per vanificare lo sforzo fatto in precedenza per dare al tutto una struttura coerente e flessibile. Sempre per lo stesso motivo, vedremo come modificare la procedura **CloseSafelyWindow()** per renderla del tutto indipendente dal programma che la chiama.

Per finire, continueremo la nostra carrellata sui coimandi dell'*AmigaDOS 1.3* con altri cinque comandi, a partire da **dir**.

Il campo MutualExclude

Due voci si dicono mutualmente esclusive se non possono essere selezionate contemporaneamente. Tale tecnica si usa per quegli attributi che siano non compatibili l'uno con l'altro. Supponiamo ad esempio di aver scritto un programma per disegnare. Uno dei menu serve a definire lo stato della penna. Tra le varie voci ce n'è una relativa al colore della penna stessa. Questo menu avrà un sottomenu le cui voci corrispondono ai vari colori disponibili. Selezionando un colore si informa il programma che la penna deve scrivere in quel colore. È evidente a questo punto che:

1. non è possibile avere due colori selezionati allo stesso tempo
2. almeno un colore deve sempre essere selezionato.

Il primo punto si ottiene appunto utilizzando il campo **Mutual Exclude**, come vedremo tra poco: il secondo si ottiene, assumendo che la prima caratteristica sia già stata abilitata, usando solo la costante **CHECKIT** nel campo **Flags** relativo agli elementi del sottomenu *Colori*, senza cioè aggiungere **MENUTOGGLE**. Vedremo tra un attimo perché.

Il campo **Mutual Exclude** è formato da quattro byte, cioè da trentadue bit. Ad ogni bit corrisponde uno dei primi trentadue elementi di un menu (o di un sottomenu). Da questo se ne deducono due cose: primo, che le voci da impostare mutualmente esclusive devono appartenere *allo stesso menu o sottomenu*; secondo, che esse devono essere tra le prime trentadue voci di quella lista. Mentre quest'ultimo punto non pone troppi problemi, dato che una lista non più di trentadue voci non indica certo un dise-

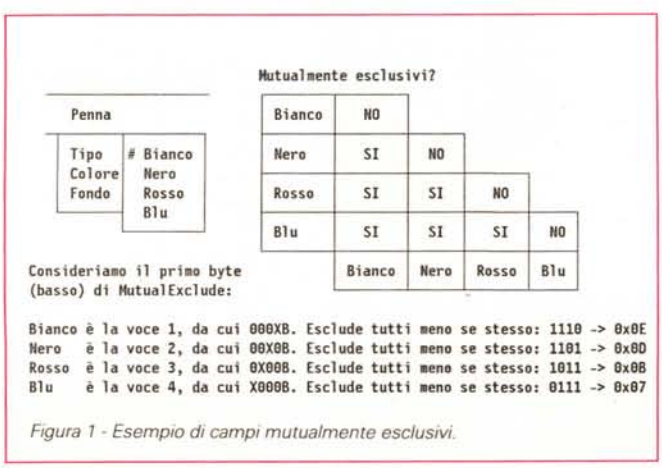

Figura 1 - Esempio di campi mutualmente esclusivi.

gno oculato della nostra struttura a menu, e che comunque anche in questo caso basta spostare gli elementi interessati in cima alla lista (non credo sia verosimile che essi siano più di questo limite massimo), il primo punto potrebbe porre qualche limite al disegno dei nostri menu. In genere, però, sembra ragionevole pensare che attributi mutualmente esclusivi facciano parte di una stessa lista. Ad ogni modo, il bit meno significativo (**«0»**) del campo **MutualExclude** corrisponde alla prima voce nella lista, il bit più significativo, quello cioè più a sinistra (**«31»**), corrisponde alla trentaduesima. Supponiamo ora che la terza voce sia mutualmente esclusiva con la prima e con la quinta. per indicare ciò , basterà impostare ad uno nel campo **MutualExclude** della terza voce i bit corrispondenti alle due voci con le quali essa è mutualmente esclusiva, cioè il primo (**«0»**) ed il quinto («4»). Analogamente queste due voci avranno il terzo bit, cioè il numero **«2»**, impostato ad uno. Tutti gli altri bit saranno nulli.

Vediamo ora come questo si applica al caso del menu dei colori, facendo riferimento alla figura 1. Ogni colore esclude tutti gli altri, tranne ovviamente se stesso. Questo vuol dire che il suo campo **MutualExclude** avrà tutti **«1»** tranne il bit corrispondente alla propria posizione nel menu. Da qui ne conseguono i valori riportati in figura. Se a questo punto impostiamo **CHECKIT** nel campo **Flags**, in modo da far sì che l'utente possa selezionare il colore desiderato facendo click con il mouse sulla voce corrispondente, ma *non* impostiamo **MENUTOGGLE**, in modo da impedire all'utente di deselezionare tale colore con un secondo click, ne risulta che l'unico modo per deselezionare un colore è quello di selezionarne un altro. Naturalmente, all'apertura del menu, il programma stesso avrà provveduto a selezionare un colore di default impostando per esso *anche* il

```
/*
**  Costanti
*/
 #define EXCLUDEALL  0xFFFFFFFF
 #define EXCLUDENONE 0x00000000

/*****************************************************************
** SetExclude: fa si che in un set contiguo di voci di una lista, ogni    **
**             voce escluda tutte le altre.                               **
*****************************************************************/
 void SetExclude(ilist,fromitem,toitem)
   ITEM         *ilist;      /* Puntatore alla lista delle voci          */
   int           fromitem;   /* Dalla voce...                            */
   int           toitem;     /* ...alla voce.                            */
 {
   int i;
   ITEM *iptr;
   LONG exclmask = EXCLUDENONE;

   toitem++;

   for (i=fromitem, iptr=ilist+fromitem; i < toitem && iptr != NULL; i++)
   {
     exclmask |= (1 << i);        /* Crea una maschera con tutti 0 tranne */
     iptr = iptr->NextItem;       /* che per il set contiguo specificato  */
   }
   for (i=fromitem, iptr=ilist+fromitem; i < toitem && iptr != NULL; i++)
   {

     iptr->MutualExclude = exclmask ^ (1 << i); /* Metti a 0 solo il bit  */

     iptr = iptr->NextItem;                     /* relativo a questa voce */
   }
 }

/*****************************************************************
** ClearExclude: cancella tutte le specifiche di mutua esclusione relati- **
**               ve ad un set contiguo di voci.                           **
*****************************************************************/
 void ClearExclude(iptr,fromitem,toitem)
   ITEM         *iptr;       /* Puntatore alla lista delle voci          */
   int           fromitem;   /* Dalla voce...                            */
   int           toitem;     /* ...alla voce.                            */
 {
   int i;

   toitem++;

   for (i = fromitem; i < toitem || iptr != NULL; i++)
   {
     iptr->MutualExclude = EXCLUDENONE;
     iptr = iptr->NextItem;
   }
 }
```

Figura 2 - SetExclude() e Clear Exclude().

```
/*****************************************************************
** BuildMenus: Costruisce i menù                                          **
*****************************************************************/
 void BuildMenus()
 {
  /*
  **  Definiamo i menù PER PRIMI
  */
   SetupMenu(&menulist[MENU_100],NULL /* nessuno */ ,
             MENU_1TX,itemlist[MENU_100]);
   SetupMenu(&menulist[MENU_200],&menulist[MENU_100],
             MENU_2TX,itemlist[MENU_200]);
   SetupMenu(&menulist[MENU_300],&menulist[MENU_200],
             MENU_3TX,itemlist[MENU_300]);

  /*
  **  Definiamo le voci PER SECONDE (attenzione all ordine, ora!)
  */
   SetupItemList(NULL,itemlist[MENU_100],ITEM_1NM,ITEM_1FL,
       itemtext[MENU_100], itemname[MENU_100],subilist[MENU_100]);
   SetupItemList(NULL,itemlist[MENU_200],ITEM_2NM,ITEM_2FL,
       itemtext[MENU_200], itemname[MENU_200],subilist[MENU_200]);
   SetupItemList(NULL,itemlist[MENU_300],ITEM_3NM,ITEM_3FL,
       itemtext[MENU_300], itemname[MENU_300],NULL              );

  /*
  **  Definiamo le sottovoci PER TERZE (attenzione all'ordine, ora!)
  */
   SetupItemList(&itemlist[MENU_100][ITEM_140],subilist[MENU_100][ITEM_140],
       SUBI_14N,SUBI_14F,subitext[MENU_100][ITEM_140],
       subiname[MENU_100][ITEM_140],NULL);
   SetupItemList(&itemlist[MENU_200][ITEM_230],subilist[MENU_200][ITEM_230],
       SUBI_23N,SUBI_23F, subitext[MENU_200][ITEM_230],
       subiname[MENU_200][ITEM_230],NULL);

  /*
  **  Impostiamo le voci mutualmente esclusive. Nel nostro caso sono la
  **  seconda e la terza del terzo menù.
  */
   SetExclude(itemlist[MENU_300],ITEM_320,ITEM_330);

  /*
  **  Fatto! E adesso associamo il tutto alla finestra.
  */
   SetMenuStrip(w,&menulist[0]);
   SaveFlags = w->IDCMPFlags;
   ModifyIDCMP(w,SaveFlags|MENUFLAGS);
   mask |= MSK_MST;
 }
```

Figura 3 - Uso della SetExclude().

valore **CHECKED**. E adesso un piccolo quiz. Supponiamo che, come in figura, il sottomenu dell'esempio contenga solo quattro colori. Intuition andrà allora a verificare solo i primi quattro bit del campo **MutualExclude**, ignorando i rimanenti. Dato che tuttavia questo campo ha comunque 32 bit, quale è la scelta migliore per gli altri, **«0»** oppure **«1»**? Premetto che tale scelta non influisce in alcun modo sul comportamento di Intuition, e che in ogni caso essa deve essere valutata caso per caso, non può cioè essere generalizzata. La risposta nella prossima puntata.

Il programma scheletro

E vediamo ora come, nel programma scheletro, abbiamo affrontato il problema della mutua esclusione delle voci.

Dato che la definizione di quali voci siano mutualmente esclusive e quali no non può essere generalizzato a meno di non pagare un prezzo troppo elevato in termini di codice, e dato che in genere solo un numero ristretto di voci ha tali caratteristiche, ho pensato di scrivere, a titolo di esempio, due piccole procedure che possono rivelarsi utili nella maggior parte dei casi (vedi figura 2).

La prima, **SetExclude()**, serve a far sì che in un set contiguo di voci, ogni voce escluda mutualmente tutte le altre. In pratica può messere usata per dire ad Intuition: *«Nel menu pincopallino, rendi mutualmente esclusive tutte le voci dalla terza alla settima incluse»*. Infatti i parametri da passare a questa funzione sono tre: il primo è il puntatore al menu (o sottomenu) a cui appartiene il set di voci interessate; il secondo ed il terzo sono rispettivamente gli identificativi della prima e dell'ultima voce dell'insieme *mutualmente esclusivo*. Se diamo un'occhiata al codice, vediamo che esso è formato da due blocchi. Nel primo viene creata una maschera di quattro byte in cui tutti i bit sono a zero tranne quelli relativi alle voci che fanno parte del set in questione. Nel secondo questa maschera viene assegnata al campo **MutualExclude** di ognuna delle voci del

Figura 4
SetupItemList().

```
/*****************************************************************************
** SetupItemList: costruisce una lista di voci                              **
*****************************************************************************/
void SetupItemList(ilink,ilist,itemnum,itemflags,it,itemname,slist)
  ITEM    *ilink;           /* Voce padre se sottovoci, se no NULL       */
  ITEM    *ilist;           /* Lista delle voci: vettore                 */
  int     itemnum;          /* Numero di voci nella lista                */
  USHORT  itemflags;        /* Caratteristiche dell'elemento             */
  ITXT    *it;              /* Vettore di strutture IntuiText da usare   */
  IDESC   *itemname;        /* Titoli delle voci della lista e comandi   */
  ITEM    *slist[];         /* Puntatore alle liste delle sottovoci      */
{
  int i, altc = 0;          /* Contatori: voci e testi alternativi       */
  SHORT   itemwidth = 0;    /* Larghezza dell'elemento                   */
  BOOL    anycmd = FALSE;   /* Tiene traccia della presenza di comandi   */
  ITXT   *alttext;          /* Puntatore alla lista dei testi alternati  */

  alttext = it + itemnum;   /* ->spazio allocato per i testi alternativi */

  for (i = 0; i < itemnum; i++)
  {
    ilist[i].NextItem    = &ilist[i+1];  /* Lega alla voce successiva    */
    ilist[i].LeftEdge    = 0;            /* Spostamento da sinistra      */
    ilist[i].TopEdge     = HITEM * i;    /* Distanza dal bordo superiore */
    if (ilink != NULL)
    {
      ilist[i].LeftEdge += ilink->Width - WDELTA;    /* >0 per sottovoci */
      ilist[i].TopEdge  -= 2*(ilink->TopEdge/HITEM); /* negativo variab. */
    }
    ilist[i].Height      = HITEM-2;      /* Altezza dell'elemento        */
    ilist[i].Flags       = itemflags;    /* Caratteristiche della voce   */
    ilist[i].MutualExclude = EXCLUDENONE; /* Tutti indipendenti, per ora */
    ilist[i].Command     = itemname[i].cmd; /* Eventuali comandi         */
    if (slist != NULL)                   /* C'è una lista di sottovoci?  */
      ilist[i].SubItem   = slist[i];     /* Sì! Puntatore alle sottovoci */
    else
      ilist[i].SubItem   = NULL;         /* No! Nessuna lista.           */
    ilist[i].NextSelect  = MENUNULL;     /* Per le selezioni multiple    */

    /*
    **  Attiva e visualizza il comando "scorciatoia"
    */
    if (ilist[i].Command != '\0')
    {
      ilist[i].Flags |= COMMSEQ;
      anycmd = TRUE;              /* C'è almeno un comando nella lista    */
    }
    /*
    **  Cloniamo la struttura base ed assegnamo al campo "IText" il titolo
    **  della voce. Se è previsto un "checkmark", lasciamo suffiente spazio
    **  a sinistra.
    */
    it[i]                = basetext;
    it[i].IText          = itemname[i].txt;
    it[i].LeftEdge      += ((itemflags & CHECKIT)  ? CHECKWIDTH : 0);
    ilist[i].ItemFill    = (APTR)&it[i];

    /*
    **  Cerca la larghezza massime tra quelle delle singole voci
    */
    itemwidth  = max( itemwidth, (ITXTL(&it[i]) +
     ((itemflags & CHECKIT)  ? CHECKWIDTH : 0)  ));    /* Marcatore?     */

    /*
    **  Se c'è un testo alternativo, assegna il campo SelectFill
    */
    if (itemname[i].alt != NULL)      /* Voce con testo alternativo       */
    {
      alttext[altc]           = basetext;
      alttext[altc].IText     = itemname[i].alt;
      alttext[altc].LeftEdge += ((itemflags & CHECKIT)? CHECKWIDTH:0);
      ilist[i].SelectFill     = (APTR)&alttext[altc];

      /*
      **  Assicurati che solo HIGHIMAGE sia impostato per questa voce
      */

      ilist[i].Flags         &= ~(HIGHBOX|HIGHCOMP);

      ilist[i].Flags         |= HIGHIMAGE;

      /*
      **  Cerca la larghezza massime tra quelle delle singole voci
      **  Un testo alternativo può infatti essere più lungo del più
      **  lungo testo normale.
      */
      itemwidth  = max( itemwidth, (ITXTL(&alttext[altc]) +
      ((itemflags & CHECKIT)  ? CHECKWIDTH : 0)  ));     /* Marcatore?  */

      altc++;                     /* Incrementiamo di uno il contatore    */
    }
    else
    {
      ilist[i].SelectFill     = NULL;
    }
  }
  ilist[itemnum-1].NextItem = NULL;     /* Ultimo elemento                */

  /*
  **  Usa come larghezza della lista delle voci, la massima trovata.
  **  Se c'è anche un solo comando, allarga il tutto.
  */
  if (anycmd) itemwidth += 2*COMMWIDTH;
  for (i = 0; i < itemnum; i++) ilist[i].Width = itemwidth + WDELTA;
}
```

```
/*************************************************************************
** TotTextNumber: conta il numero totale di ITXT da allocare           **
*************************************************************************/
 int TotTextNumber(inamptr,descnum)
   IDESC        *inamptr;    /* Puntatore al vettore dei descrittori     */
   int           descnum;    /* Numero di descrittori nel vettore        */
 {
   int i, totnum;
   for (i = 0, totnum = descnum; i < descnum; i++)
   {
     if (inamptr[i].alt != NULL) totnum++;
   }
   return (totnum);
 }
```

Figura 5 - TotTextNumber().

```
/*
**  Testi (voci & sottovoci)
*/
 itemtext[MENU_100] = (ITXT *)AllocRemember(&rememory,
              TotTextNumber(itemname[MENU_100],ITEM_1NM) * sizeof(ITXT),
              MEMF_CLEAR);
 if (itemtext[MENU_100] == NULL) CloseAll();
 itemtext[MENU_200] = (ITXT *)AllocRemember(&rememory,
              TotTextNumber(itemname[MENU_200],ITEM_2NM) * sizeof(ITXT),
              MEMF_CLEAR);
 if (itemtext[MENU_200] == NULL) CloseAll();
 itemtext[MENU_300] = (ITXT *)AllocRemember(&rememory,
              TotTextNumber(itemname[MENU_300],ITEM_3NM) * sizeof(ITXT),
              MEMF_CLEAR);
 if (itemtext[MENU_300] == NULL) CloseAll();


 subitext[MENU_100][ITEM_110] = (ITXT *)NULL;          /* per sicurezza */
 subitext[MENU_100][ITEM_120] = (ITXT *)NULL;          /* per sicurezza */
 subitext[MENU_100][ITEM_130] = (ITXT *)NULL;          /* per sicurezza */
 subitext[MENU_100][ITEM_140] = (ITXT *)AllocRemember(&rememory,
      TotTextNumber(subiname[MENU_100][ITEM_140],SUBI_14N) * sizeof(ITXT),
      MEMF_CLEAR);
 if (subitext[MENU_100][ITEM_140] == NULL) CloseAll();
 subitext[MENU_100][ITEM_150] = (ITXT *)NULL;          /* per sicurezza */

 subitext[MENU_200][ITEM_210] = (ITXT *)NULL;          /* per sicurezza */
 subitext[MENU_200][ITEM_220] = (ITXT *)NULL;          /* per sicurezza */
 subitext[MENU_200][ITEM_230] = (ITXT *)AllocRemember(&rememory,
      TotTextNumber(subiname[MENU_200][ITEM_230],SUBI_23N) * sizeof(ITXT),
      MEMF_CLEAR);
 if (subitext[MENU_200][ITEM_230] == NULL) CloseAll();
```

Figura 6 - Start All().

```
/*************************************************************************
** CloseSafelyWindow: chiude una finestra che condivide con altre la   **
**                    stessa porta utente nel modo più sicuro          **
**                    Per l'ultima usa invece CloseWindow().           **
*************************************************************************/
 void CloseSafelyWindow(wptr,fptr)
   struct Window *wptr;
   struct TextFont *fptr;
 {
   IMSG *scan;
   struct MsgPort *up;

  /*
  **  Si assume di aver già cancellato la barra menù relativa alla finestra
  */
   if (wptr == NULL) return; /* Logica della scatola nera: per sicurezza  */

  /*
  **  Dato che stiamo lavorando su una lista di sistema è necessario
  **  disabilitare temporaneamente il multitasking.
  */
   Forbid();               /* Blocca il multitasking.                     */

  /*
  **  Ciclo sulla lista dei messaggi arrivati alla porta utente. Quelli
  **  relativi alla finestra da chiudere sono rimossi.
  */
   up = wptr->UserPort;
   for (scan = (IMSG *)up->mp_MsgList.lh_Head;      /* Inizio lista       */
        (IMSG *)scan->ExecMessage.mn_Node.ln_Succ;  /* Successivo è NULLO */
        scan = (IMSG *)scan->ExecMessage.mn_Node.ln_Succ)  /* Successivo  */
   {
     if (scan->IDCMPWindow == wptr) /* Il messaggio è per questa finestra */
     {
      /*
      **  Usa Remove() invece di GetMsg() dato che quest'ultima toglie
      **  SEMPRE il messaggio in testa alla lista.
      */
       Remove((NODE *)scan);   /* OK. Cancelliamolo dalla coda messaggi.. */
       ReplyMsg((EMSG *)scan); /* e rispondiamo al mittente.              */
     }
   }

   wptr->UserPort = NULL;  /* Evita che Intuition deallochi la porta ut.  */
   ModifyIDCMP(wptr,NULL); /* OK. Adesso effettuiamo la chiusura "logica" */

   Permit();               /* Ripristina il multitasking.                 */

  /*
  **  OK. Ora possiamo chiudere tranquillamente la finestra
  */
   CloseWindow(wptr);
   if (fptr) CloseFont(fptr); /* Nel caso avvessimo caricato un font      */
 }
```

Figura 7 - GloseSafelyWindow().

set, avendo cura di mettere prima a zero il bit relativo alla voce interessata, dato che una voce non può escludere se stessa. È importante comprendere come questo sia solo un esempio. Molte altre scelte potevano essere fatte. Ad esempio, invece di scrivere

iptr→MutualExclude = exclamask (1<< i);

potevamo scrivere

iptr→MutualExclude |= exclamask (1<< i);

in modo da non salvaguardare eventuali altre esclusioni impostante in precedenza per quella singola voce.

La seconda funzione, **ClearExclude()**, cancella tutte le specifiche di mutua esclusione relative ad un set contiguo di voci. I parametri passati sono gli stessi usati in **SetExclude()**. Anche in questo caso si poteva pensare di azzerare solo quei bit interessati dallo stesso insieme di voci, piuttosto che semplicemente azzerare il campo **MutualExclude** per ogni voce dell'insieme. Una variazione di questo tipo può facilmente essere implementata sulla falsalinea della funzione precedentemente descritta, cioè creando prima una maschera opportuna, e poi utilizzando quest'ultima sul campo da modificare, in modo da non modificare i bit non interessati. Per l'occasione ho definito due nuove costanti, **EXCLUDEALL** che rappresenta la mutua esclusione di tutte e trentadue le prime voci di un menu, e **EXCLUDENONE**, che equivale a non avere alcuna specifica di esclusione. In particolare quest'ultima costante l'ho utilizzata anche nella procedura **SetupItemList()** (vedi figura 4) al posto della costante generica **NULL**, tanto per dare una certa coerenza al tutto.

```
/*
**  Prototipi delle nuove funzioni di servizio
*/
 int  TotTextNumber( IDESC*, int );
 void SetExclude   ( ITEM *, int, int );
 void ClearExclude ( ITEM *, int, int );
```

Figura 8 - Prototipi delle nuove funzioni.

Usiamo ora la **SetExclude()** per definire come mutualmente esclusive la seconda e la terza voce del terzo menu.

Questo va fatto nella **BuildMenus()** dopo aver chiamato le funzioni **SetupMenu()** e **SetupItemList()**, ma prima di chiamare la **SetMenuStrip()**, come riportato in figura 3.

I testi alternati

Come certamente ricorderete, nella scorsa puntata abbiamo aggiunto alla procedura **SetupItemList()** la gestione del campo **SelectFill** per supportare i testi alternati delle voci e delle sottovoci. Nel far ciò, allo scopo di evitare di allocare inutilmente una struttura **IntuiText** in più per ogni voce nella **StartAll()** dato che in genere solo un numero limitato di voci hanno testi alternati, avevamo dato alla stessa **SetupItemList()** la responsabilità di effettuare tale allocazione per le sole voci interessate. Questo però comporta una deviazione dal proposito di fare di questa funzione una funzione di servizio, secondo cioè la logica della *scatola nera*. Come fare allora?

Per risolvere il problema ho scritto una nuova funzione, la **TotTextNumber()**, riportata in figura 5. Questa funzione ha lo scopo di contare in ogni menu o sottomenu il numero di testi alternativi effettivamente desiderati, e di restituire il numero totale di strutture **intuiText** (o

La scheda tecnica

Continua la nostra carrellata sui comandi dell'*AmigaDos 1.3*.

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:	DISKCHANGE
Formato:	DISKCHANGE <unità>
Sintassi:	DISKCHANGE "DRIVE/A"
Scopo:	Informa l'Amiga che è stato cambiato un discetto in una unità da 5"1/4
Specifiche:	L'Amiga non è in grado di sapere se è stato cambiato o menu un un dischetto in una unità da 5"1/4. Per questo, sia che ciò sia avvenuto, sia nel caso che si sia rinominato un dischetto già inserito con il comando RELABEL, è necessario informare il sistema con questo comando.
Esempio:	DISKCHANGE df3:

Comando:	DISKDOCTOR
Formato:	DISKDOCTOR <unità>
Sintassi:	DISKDOCTOR "DRIVE/A"
Scopo:	Cerca di riparare un disco danneggiato
Specifiche:	Cerca di riparare un disco danneggiato, quel tanto almeno che permetta di copiare i file recuperati su di un altro dischetto. Lavora sia con lo SFS, che con il FFS. Da notare che, per usare DISKDOCTOR con il FFS, il campo DOSType nella MOUNTLIST corrispondente all'unità in questione, deve avere come valore 0x444F5301, altrimenti si rischia di danneggiare definitivamente i dati sul disco. DISKDOCTOR verifica la memoria disponibile prima di operare, e cambia il blocco di partenza nel tipo DOS.
Esempio:	DISKDOCTOR df1:

Comando:	DIR
Formato:	DIR [<nome>] [OPT A|I|AI|D] [ALL] [DIRS] [INTER] [FILES] interattivo: [B|BACK] | [DEL|DELETE] | [E|ENTER] | [Q|QUIET] | [T|TYPE] | [C|COM [<comando>]]
Sintassi:	DIR "NAME,OPT/K,ALL/S,DIRS/S,INTER/S,FILES/S" interattivo: "B=BACK/S,DEL=DELETE/S,E=ENTER/S,Q=QUIT/S,T|TYPE, C=COM/S with COMMAND,C=COM/K"
Scopo:	Visualizza il contenuto ordinato di un direttorio
Specifiche:	Se non è specificata alcuna opzione, visualizza tutti i sotto-direttorii di primo livello ed i file relativi al direttorio specificato, od a quello corrente, se <nome> viene omesso. I direttori sono listati prima dei file, ed entrambi sono ordinati in ordine alfabetico. Le opzioni sono: * ALL | OPT A visualizza i sotto-direttori di TUTTI i livelli con i loro file, oltre ai file del direttorio specificato * DIRS | OPT D visualizza solo i sotto-direttori del primo livello * INTER | OPT I entra in modo INTERATTIVO, permette cioè di lanciare una delle seguenti azioni dopo aver visualizzato un file od un direttorio. Vedi sotto per le caratteristiche di questo modo. * OPT AI come OPT A più OPT I * FILES visualizza SOLO i file del direttorio --- Il modo interativo -- caratteristiche ufficiali Quando DIR entra in modo interattivo, visualizza ogni elemento scandito seguito da un punto interrogativo e si mette in attesa di una qualche azione da parte dell'utente: * <CR> a vuoto continua la scansione * C | COM chiede all'utente di digitare un comando che poi esegue, indipendentemente dall'elemento * C | COM "cmd" come sopra, ma esegue subito il comando tra virgolette * E | ENTER se l'elemento visualizzato è un direttorio, visualizzane il contenuto in modo interattivo * B | BACK torna a visualizzare il direttorio di livello superiore o termina se non c'è * DEL | DELETE cancella l'elemento (file o direttorio vuoto) * T | TYPE se l'elemento visualizzato è un file, allora visualizzane il contenuto. Ctrl-C per tornare al modo interattivo * Q | QUIT termina il modo interattivo * ? dà la lista dei comandi in modo interattivo
Esempio:	DIR RAM: ALL clipboards (dir) env (dir) c (dir) run eco ls type shell-startup t (dir) Command-00-T01 disk.info

ITXT secondo la convenzione adottata qualche puntata fa) da allocare, incluse quelle relative ad i testi base. Per far ciò è necessario passare come parametri il puntatore al descrittore del menu, al vettore cioè che viene utilizzato anche dalla **SetupItemList()** per generare le strutture da passare ad Intuition, ed il numero di elementi in esso contenuti, cioè il numero di voci per quel menu.

Questa funzione viene poi utilizzata nella **StartAll()** come si può vedere in figura 6, sostituendo ai moltiplicatori **ITEM_xNM** e **SUBI_xyN** della sizeof (ITXT), rispettivamente

TotTextNumber(itemname[MENU_x00],ITEM_xNM)

e

TotTextNumber(subiname[MENU_x00][ITEM_xy0], SUBI_xyN)

Naturalmente questa soluzione ha richiesto dei cambiamenti massicci nella **SetupMenuList()**. Vediamo quali.

Innanzi tutto, per aumentare la leggibilità del codice, ho ridefinito **IntuiTextLength()** come **ITXTL()**. Quindi ho rinominato il puntatore alla lista dei testi alternati **alttext**. Quest'ultima non viene più allocata dinamicamente dalla stessa **SetupItemList()**, essendo già stata allocata implicitamente grazie alla **TotTextNumber()**. Di conseguenza basta definire tale puntatore come un numero di strutture **IntuiText** equivalente ad **itemnum** a partire dal puntatore alla lista di *tutte* le strutture testo, sfruttando la ben nota proprietà dei puntatori C di venir incrementati dai numeri interi di tanti byte quanti sono quelli che compongono l'elemento a cui essi puntano.

A questo punto il gioco è fatto. Avendo infatti a disposizione un'area di memoria sufficiente a contenere le strutture **IntuiText** sia dei testi base che di quelli alternati, ed avendo a disposizione sia il puntatore all'intera area, sia quello alla zona riservata ai testi alternati, la **SetupItemList()** non deve far altro che utilizzare quest'ultimo ogni volta che ha bisogno di una struttura per un testo alternato, incrementando opportunamente di uno il contatore corrispondente **Alt**. Il risultato è quello di aver eliminato il secondo ciclo aggiunto la volta scorsa, semplificando così notevolmente il codice, e di aver ricondotto la procedura che costruisce la lista delle voci al suo ruolo di funzione di servizio.

CloseSafelyWindow()

La stessa cosa può essere fatta anche per l'altra funzione *«sporca»* tra quelle nel file **sklprocs.c.** Qui la cosa è più semplice, come si può vedere in figura 7. Basta infatti ricavare il puntatore alla porta messaggi da quello relativo alla finestra da chiudere, senza dover scomodare così una variabile globale.

Per concludere, in figura 8 sono riportati i prototipi delle nuove funzioni introdotte in questa puntata.

Conclusione

Nella prossima puntata vedremo come far gestire al nostro programma i vari messaggi di errore ed informativi, introducendo un nuovo oggetto alla nostra collezione: il quadro automatico *[Automatic Requester]*. Nel frattempo esercitatevi con i segnalatori per le voci mutualmente esclusive e provate a modificare le due funzioni introdotte in questa puntata in modo da non cancellare le specifiche di mutua esclusione impostate in precedenza.

Buon lavoro!

Comando:	ECHO
Formato:	ECHO <stringa> [NOLINE] [FIRST <n>] [LEN <l>]
Sintassi:	ECHO "STRING,NOLINE/S,FIRST/S,LEN/S"
Scopo:	Visualizza a terminale una stringa
Specifiche:	Visualizza a terminale una stringa. Questa può essere compresa tra virgolette o meno. Se compresa tra virgolette, le seguenti sequenze di controllo sono riconosciute: *N nuova linea *E carattere ESC *E[carattere CSI ** carattere * *" carattere " altrimenti vengono prese come sono. In genere ECHO mette alla fine della stringa un carattere di SALTO A NUOVA LINEA. Le opzioni sono: * NOLINE non saltare a nuova linea a fine stringa * FIRST <n> parti a visualizzare la stringa dal carattere n-simo * LEN <l> lunghezza della stringa da visualizzare a partire dal carattere n-simo se FIRST è specificato, altrimenti visualizza gli ULTIMI <l> caratteri.
Esempio:	ECHO "Oggi è martedì" FIRST 8 visualizza martedì --- ECHO "Oggi è martedì" FIRST 8 LEN 5 visualizza marte --- ECHO "Oggi è martedì" LEN 2 visualizza dì

Comando:	ENDCLI
Formato:	ENDCLI
Sintassi:	ENDCLI
Scopo:	Termina un processo CLI interattivo o una Shell
Specifiche:	Può essere utilizzato per terminare sia un processo CLI interattivo, sia un processo di tipo Shell.

I seguenti comandi non hanno subito variazioni nel passaggio dalla versione 1.2 alla 1.3, e per questo non sono riportati in questa scheda:

BREAK	ELSE	LAB	RENAME
CD	ENDIF	MAKEDIR	SORT
ED	FAILAT	QUIT	STACK
EDIT	FAULT	RELABEL	WHY