Programmare in C su Amiga (25)

di Dario de Judicibus
(MC2120)

Bottoni, cursori, leve, caselle... Se una volta lavorare con un elaboratore voleva dire intrecciarsi le dita su una tastiera, introducendo i dati uno per uno a manina, aspettando pazientemente la risposta a terminale, oggi le nuove tecnologie hardware e software ci introducono in un mondo di grafica e strumenti specializzati sempre più fantascientifici, quali track-ball, tavolette grafiche, hand-scanner, mentre al posto del vecchio schermo a fosfori verdi ci ritroviamo in un caleidoscopio di immagini e colori che si muovono e cambiano aspetto al tocco più o meno sapiente dell'utente che, come un direttore di orchestra, agita con sicurezza la bacchetta del 2000: il mouse. Poetico, non è vero? Forse per l'utente, ma tutto ciò bisogna anche implementarlo. Vediamo un po' come stanno le cose dalla parte del programmatore...

Introduzione

Una volta, quando ancora non c'erano le interfacce grafiche, i programmi interagivano con l'utente per mezzo di un meccanismo del tipo *domanda/risposta*. In pratica il programma visualizzava a terminale una domanda, come ad esempio:

```
Introduci le coordinate nel formato X, Y, Z
oppure premi INVIO per terminare:
```

e l'utente immetteva i dati come richiesto. Allo stesso modo il programma forniva informazioni durante l'esecuzione o chiedeva conferma all'utente se effettuare o meno una data operazione. Il tutto avveniva utilizzando la tastiera come meccanismo di ingresso ed il video per visualizzare i dati in uscita. Ancora oggi molti programmi lavorano in questo modo.

Col passare del tempo, tuttavia, e grazie alla maggiore potenza disponibile in macchine che stanno tranquillamente sulla scrivania di un ufficio, si sono incominciati ad affermare nuovi meccanismi di interazione tra l'applicazione e l'utente. Come era già successo con i linguaggi di alto livello (C, Pascal, Basic,...), che avevano spostato l'interfaccia con la macchina verso il programmatore, permettendogli di lavorare in termini di concetti piuttosto che di istruzioni di linguaggio macchina, anche nel campo delle interfacce tra applicazione ed utente la tendenza è stata di spostare il tutto verso quest'ultimo, rendendo così i prodotti più intuitivi e facili da usare. Si sono così sviluppate delle interfacce che cercano di simulare il mondo che ci circonda. Esse si basano sul concetto di *oggetto*, rappresentato graficamente da una piccola immagine chiamata *icona*, che rappresenta appunto un oggetto che già l'utente conosce nel mondo reale il quale ha un comportamento analogo a quello del programma che sta dietro all'oggetto software *[object-based interfaces]*.

Questo ha richiesto tuttavia una ridefinizione delle modalità di interazione con l'utente, che ha portato a sviluppare nuovi elementi software ed hardware destinati a formare il nuovo ambiente di lavoro sia nell'informatica individuale, che in quella dei grossi sistemi. Un esempio classico è il mouse.

Una interfaccia grafica, come è quella basata su Intuition, deve prevedere quindi tutta una serie di oggetti software che vanno a sostituire il vecchio dialogo *domanda/risposta* e che difatto offrono all'utente una gamma di possibilità molto più vasta del metodo precedente. La maggior parte di questi oggetti sono stati definiti per analogia con i sistemi di controllo degli apparecchi elettrici che oramai hanno invaso tutte le case, e che quindi risultano sufficientemente intuitivi anche per chi si avvicina ad un elaboratore per la prima volta. Tali oggetti prendono nomi spesso molto differenti a seconda del sistema operativo (Amiga, OS/2, NeXT, X-Windows). In Amiga si chiamano *gadget*, ma noi li chiameremo, in italiano, *controlli*, dato che la loro funzione è quella di permettere all'utente di controllare in una certa misura il funzionamento del programma.

Da questa puntata incominceremo a

Figura 1 - IntuiText.

```
struct IntuiText
{
        UBYTE     FrontPen   ;  /* Penna per il carattere                */
        UBYTE     BackPen    ;  /* Penna per lo sfondo                   */
        UBYTE     DrawMode   ;  /* Modo grafico                          */
        SHORT     LeftEdge   ;  /* Posizione del testo da sinistra       */
        SHORT     TopEdge    ;  /* Posizione del testo dall'alto         */
  struct TextAttr *ITextFont ;  /* Formato, stile, dimensioni, ...       */
        UBYTE     *IText     ;  /* Il testo da visualizzare              */
  struct IntuiText *NextText ;  /* Successiva struttura nella catena     */
};
```

Figura 2 - TextAttr.

```
struct TextAttr
{
        STRPTR    ta_Name    ;  /* Nome del formato [font]               */
        UWORD     ta_YSize   ;  /* Altezza del formato                   */
        UBYTE     ta_Style   ;  /* Stile del formato                     */
        UBYTE     ta_Flags   ;  /* Caratteristiche varie                 */
};
```

parlare di controlli *[gadget]* e di tutte quelle strutture ad essi collegate. Lasceremo da parte quindi per un po' il nostro programma scheletro, che ci ha accompagnato in questi ultimi mesi, per poi riprenderlo tra qualche puntata per cercare di applicare quanto fin qui imparato sui controlli che Intuition mette a disposizione del programmatore.

Prima di incominciare però con i controlli veri e propri, parleremo di tre strutture che verranno usate molto spesso nella definizione di tali oggetti, una delle quali abbiamo già incontrato in passato, senza averla tuttavia analizzata in dettaglio, e cioè la struttura **IntuiText**. Le tre strutture in questione sono quindi:

IntuiText che serve a definire testi in formati, dimensioni e stili differenti (l'avevamo già incontrata quando abbiamo parlato di menu);

Border che permette di definire dei contorni poligonali utilizzando segmenti di vario tipo, spessore, lunghezza e inclinazione;

Image che serve a definire immagini grafiche di qualunque tipo e complessità.

Tutte queste strutture hanno in comune una cosa: gli oggetti che esse rappresentano sono posizionati rispetto ad un *contenitore*, ad un altro oggetto che cioè li contiene. Questi può essere uno schermo, una finestra, un controllo od un quadro (in inglese, *screen, window, gadget,* e *requester*). La posizione dell'oggetto contenuto è in genere definita dalle coordinate relative ad un punto dell'oggetto contenitore, misurate in pixel. Di solito il punto di riferimento è l'angolo in alto a sinistra del contenitore. Chiameremo tale punto *punto origine del contenitore*, o più semplicemente *origine*.

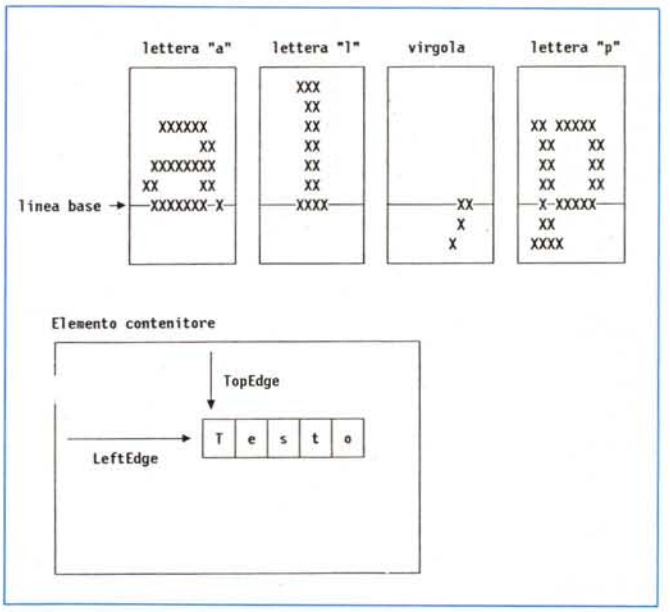
Da notare che lo stesso oggetto può essere referenziato più volte in differenti contenitori nello stesso momento, e di conseguenza visualizzarlo più volte sullo schermo. Questo permette di definire una sola volta un dato elemento anche quando il pannello che si sta componendo ne contiene più di uno.

Ovviamente, dato che la definizione è unica, se essa viene modificata, cambiano anche nello stesso modo tutti gli oggetti che si basano su di essa. Questa tecnica è molto utile qualora si debbano creare matrici di oggetti all'interno di uno stesso contenitore.

Ognuno di questi oggetti può essere utilizzato in due modi. Da una parte essi possono essere referenziati in altre strutture quali menu, controlli e quadri, e di conseguenza visualizzati quando viene chiamata la funzione che attiva tali elementi grafici. Dall'altra essi possono essere disegnati direttamente sullo schermo od in una finestra di uno schermo usando delle apposite funzioni, come ad esempio **PrintIText()**.

In questa puntata parleremo di testi (strutture **IntuiText** e **TextAttr**), mentre nella prossima entreremo in dettaglio per quello che riguarda gli elementi grafici (strutture **Border** e **Image**).

Figura 3
Il posizionamento del testo.



IntuiText

In un ambiente grafico un testo non è più definito solo dai caratteri che lo compongono, ma è necessario specificare tutta una serie di altre caratteristiche e parametri, come ad esempio la forma del carattere *[font]*, le dimensioni, lo stile e, anche se può sembrare strano, la posizione sullo schermo. Infatti, dal punto di vista di una interfaccia grafica, un testo non è altro che un elemento grafico come tutti gli altri, e va posizionato opportunamente. L'interfaccia infatti non è in grado di fare assunzioni su dove vada un certo testo, come accade nella vecchia interfaccia *a comandi*. Lì la cosa è semplice: ogni testo è separato dal precedente da un ritorno carrello, viene sempre messo una riga sotto, e sempre a partire da colonna uno. Al più si può fare qualche piccola operazione estetica indentando alcune righe rispetto ad altre. La definizione dei caratteri di controllo ANSI è stato uno dei primi tentativi di dare al programmatore un controllo maggiore sul testo stesso.

Nel caso in cui tuttavia essi non vengano specificati, tornano a valere le vecchie regole.

Nel caso di una interfaccia grafica, il controllo sul testo è molto elevato. Negli ambienti *DisplayScript* si può arrivare a curare e deformare persino i testi.

Tuttavia il prezzo da pagare è la necessità di specificare qualche informazione in più anche in quei casi che una volta venivano gestiti automaticamente.

```
void PrintIText    /* Stampa il testo nel raster in una certa posizione */
(
  struct RastPort  *RastPort  , /* Destinazione del testo              */
  struct IntuiText *IText     , /* Testo da stampare (anche una carena) */
  SHORT             TopOffset , /* Posizione dall'alto                 */
  SHORT             LeftOffset  /* Posizione da sinistra               */
);

int IntuiTextLength   /* Calcola la lunghezza in pixel del testo        */
(
  struct IntuiText *IText       /* Testo di cui calcolare la lunghezza  */
);
```

Figura 4
PrintIText() ed IntuiTextLength().

La struttura **IntuiText** (vedi figura 1) è formata da otto campi:
FrontPen è il numero del registro del colore utilizzato per tracciare i caratteri;
BackPen è il numero del registro del colore utilizzato per tracciare lo sfondo, nel caso sia stato specificato il modo grafico **JAM2**;
DrawMode è il modo grafico di tracciamento del testo, incluso lo sfondo del carattere vero e proprio;
LeftEdge indica la posizione del testo, in pixel, rispetto al lato sinistro dell'elemento che lo contiene (vedi più avanti);
TopEdge indica la posizione del testo, in pixel, rispetto al lato superiore dell'elemento che lo contiene (vedi più avanti);
ITextFont serve a definire il formato, le dimensioni e lo stile del testo, per mezzo della struttura **TextAttr**, che vedremo più avanti;
IText è il testo da visualizzare;
NextText punta ad un'altra struttura **IntuiText**, permettendo così di collegare più testi insieme a formare una lista; ogni testo può ovviamente avere caratteristiche completamente differenti.

Come già detto nella 7° puntata, apparsa nel lontano 1988, il valore assegnato alle varie penne disponibili si riferisce al registro che contiene il colore, non al colore stesso. Se si cambia la tavolozza dei colori ma non si modifica il numero del registro, il testo sarà reso nei colori che nella nuova tavolozza corrispondono ai registri specificati.

I modi grafici sono quattro (**JAM1, JAM2, COMPLEMENT** e **INVERSID**), e sono stati descritti nella 13° puntata, stampata sul numero 86 di MCmicrocomputer, quando abbiamo parlato della libreria grafica. Dato che è passato più di un anno, rivediamoli brevemente insieme.

Innanzitutto definiamo due aree grafiche relative alla visualizzazione dei caratteri: lo sfondo, su cui il carattere è tracciato, che ha la forma di un rettangolo le cui dimensioni corrispondono all'altezza ed alla larghezza del formato scelto, e il carattere vero e proprio, cioè il tratto che rappresenta il simbolo che vogliamo visualizzare. Detto questo vediamo qual è l'effetto dei vari modi grafici.

JAM1

Viene disegnato il solo carattere nel colore specificato da **FrontPen**. Lo sfondo rimane quello pre-esistente, sia che sia uniforme, sia che contenga una immagine. Si parla in questo caso di *sovra-impressione*. Nel caso di testi, questo modo è conveniente solo se il fondo ha un colore uniforme che garantisca il necessario contrasto, oppure se il carattere è sufficientemente grande da risaltare in ogni caso. A mio avviso, in quest'ultimo caso può tornare utile usare uno stile bordato *[outline]*. Ovviamente dipende anche da quanto il programmatore abbia il controllo della situazione. Se sapete con sicurezza quale sarà lo sfondo, non ci sono problemi a usare **JAM1** (ad esempio se il testo è in nero e lo sfondo è in colori pastello), ma se viceversa state scrivendo su un'immagine che il programma ha caricato dietro richiesta dell'utente, non potendo sapere a priori di che immagine si tratta sarebbe meglio usare **JAM2**.

JAM2

In questo caso viene disegnato sia il carattere nel colore specificato da **FrontPen**, sia lo sfondo, in quello specificato da **BackPen**. In questo modo lo sfondo viene reso come un rettangolo che «cammina» con il testo, e rimpiazza [replace] lo sfondo pre-esistente. Questa è una buona scelta se non sapete con che sfondo avete a che fare. Infatti, se il testo non è molto esteso e si scelgono due colori ben contrastati, possiamo avere due possibilità:
1. il fondo pre-esistente è uguale allo sfondo specificato in **BackPen**; in questo caso la visibilità è garantita dal colore scelto per il carattere;
2. il fondo pre-esistente è uguale al colore del carattere specificato in **FrontPen**; in questo caso la visibilità è garantita dal colore scelto per lo sfondo.

Lo svantaggio è che il fondo del contenitore viene cancellato dallo sfondo del testo, anche quando il carattere è uno spazio, cosa che in certi casi può risultare inaccettabile, da un punto di vista estetico, specialmente se le varie righe del testo sono di diversa lunghezza.

COMPLEMENT

Selezionando questo modo **FrontPen** e **BackPen** sono completamente ignorati. Il carattere viene tracciato usando come colore quello complementare del colore del fondo pre-esistente. Anche questa, apparentemente, potrebbe sembrare una buona scelta al fine di garantirsi una certa visibilità del testo qualunque sia il fondo. In realtà le cose non stanno proprio così. Quello che viene complementato (scambiando cioè tutti i bit **"0"** in **"1"** e viceversa), non è infatti il colore vero e proprio, ma il valore del registro di colore. Ad esempio, in una immagine a 16 colori (4 piani), il complemento al

La scheda tecnica

Scusandoci per la forzata pausa del numero scorso dovuta a mancanza di spazio, continuiamo 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:	ENDSKIP
Formato:	ENDSKIP
Sintassi:	ENDSKIP
Scopo:	Termina un blocco SKIP in una macro di sistema (script file)
Specifiche:	Se ENDSKIP viene incontrato durante l'esecuzione del comando SKIP, la macro riprende alla linea successiva alla ENDSKIP ed il segnalatore di errore viene impostato a WARN.

colore caricato nel registro 12, cioè **0×0C≡1100**, è il colore caricato nel registro **0011≡0×03**.

Ma nulla impedisce che nei due registri sia caricato *lo stesso colore*, o comunque due colori poco contrastanti fra loro! Inoltre in questo modo il testo non è visualizzato uniformemente utilizzando sempre lo stesso colore, ma questi varia col variare del fondo preesistente, cosa che può renderne pesante la lettura. Anche qui dipende ovviamente fino a che punto sapete con che fondo avete a che fare... Uno dei vantaggi di questo modo è che se riscrivete lo stesso testo, sempre specificando **COMPLEMENT**, nella stessa esatta posizione, il risultato sarà quello di cancellare il testo precedente ripristinando il fondo pre-esistente qualunque esso fosse.

INVERSID

Questo modo è particolarmente utile per i testi, e viene generalmente specificato insieme ad uno dei due modi **JAMx**. Se infatti si seleziona **JAM1|INVERSID**, lo sfondo del carattere viene reso nel colore specificato da **FrontPen**, mentre il carattere stesso è trasparente, lascia passare cioè il fondo pre-esistente. Nel caso invece di **JAM2|INVERSID**, lo sfondo del carattere utilizza **FrontPen**, mentre il carattere stesso viene tracciato con il colore relativo a **BackPen**. L'effetto risultante è quello dello stile invertito *[reverse]*.

Per vedere come si posiziona un testo, facciamo riferimento alla figura 3.

Ogni carattere va immaginato in un rettangolo. Tale rettangolo è tagliato ad una certa altezza da una linea orizzontale, chiamata *linea base [baseline]*. Su tale linea poggia la maggior parte dei caratteri ed il punto. Tutti i caratteri ed i simboli definiti in un certo gruppo *[font]* devono rimanere nei limiti del rettangolo. Alcune lettere, come la *p* o la *g* tagliano la linea base, altre come la *l* o la *t* possono arrivare a toccare il lato superiore dello stesso. In genere, quando si scrive un testo, ad esempio in un quaderno a righe, lo si scrive facendo corrispondere la linea base con la riga del quaderno, per cui alcune lettere scenderanno sotto tale riga, mentre la maggior parte dei caratteri starà sopra. Nel nostro caso, il valore di **TopEdge** va misurato a partire dal lato superiore del rettangolo che include il carattere, mentre **LeftEdge** viene misurato a partire dal lato sinistro, come mostrato appunto in figura 3.

Il testo vero e proprio viene fornito come una stringa che deve finire con il byte **0×00** *[null-terminated]*.

Come detto in precedenza, una struttura **IntuiText** può essere visualizzata sia in modo indiretto, attraverso l'attivazione di un'altra struttura che la contiene o contiene un puntatore ad essa, sia direttamente, tramite la funzione **PrintIText()**, riportata in figura 4. Tale funzione richiede di specificare il puntatore ad una struttura **RastPort** corrispondente all'elemento contenitore in cui il testo va visualizzato, quale ad esempio quello di una finestra o di uno schermo. A questa si aggiungono, oltre ovviamente al puntatore alla struttura da visualizzare, due valori che, se non nulli, vanno a sommarsi algebricamente ad i loro analoghi specificati nella struttura **IntuiText** per formare le coordinate di posizionamento del testo. Il testo risultante avrà quindi come posizione verticale:

Comando:	EVAL
Formato:	EVAL <espressione> [TO <file>] [LFORMAT=<stringa>]
Sintassi:	EVAL "EXPRESSION,TO/K,LFORMAT/K"
Scopo:	Calcola espressioni elementari
Specifiche:	EVAL serve a calcolare e stampare a terminale espressioni matematiche elementari. I due operandi possono essere numeri interi decimali, esadecimali od ottali. Un numero intero è sempre considerato decimale a meno che non abbia come prefissi 0X oppure #X che servono a specificare numeri esadecimali, o 0 oppure # che servono a specificare numeri ottali. Possono anche essere usate le parentesi nel formalismo SOA. E' anche posssibile chiedere il valore di un certo carattere incluso fra apici (ad esempio 'F'). Le operazioni supportate sono le seguenti: + addizione, & and, < sposta bit a sinistra - sottrazione, | or, > sposta bit a destra * moltiplicazione, ~ not / divisione, - negazione Il formato del risultato può essere controllato via LFORMAT. Il risultato può venire visualizzato come numero decimale (%N), esadecimale (%X), ottale (%O) oppure come un carattere (%C). La sequenza *N può essere usata per andare a capo. %X e %O richiedono di specificare la lunghezza del numero (ad esempio %X4 visualizza un esadecimale a quattro cifre).
Esempio:	EVAL (8-4)*2 LFORMAT="[0x%X2]*N" produce [0x08] EVAL (0xA&0xC)+30 LFORMAT="%C *N" produce &

Comando:	EXECUTE
Formato:	EXECUTE <comando> <argomento>...
Sintassi:	EXECUTE "COMMAND,ARGUMENTS"
Scopo:	Esegue una macro di sistema con la sostituzione dei parametri
Specifiche:	EXECUTE esegue una macro di sistema non ARexx (script file) e, se forniti, sostituisce gli argomenti ai parametri presenti nel file. Per far questo crea un file temporaneo in T: se definito, altrimenti in :T. Per una migliore gestione di processi concorrenti (multitasking), si può sfruttare la variabile interna <$$> che corrisponde al numero di CLI dal quale la macro è stata lanciata.

Comando:	FF
Formato:	FF [-0] [-N]
Sintassi:	FF "-0/S,-N/S"
Scopo:	Accelera la visualizzazione del testo
Specifiche:	FF (FastFont) è un programma della Microsmiths, Inc. che serve ad accelerare il testo visualizzato a schermo. -0 attiva il programma, -N lo disattiva. Non si può ottenere un aiuto con il punto interrogativo come con gli altri comandi. Secondo il manuale la sintassi "FF <font>" dovrebbe permettere di sostituire il font di sistema con un altro font 8x8, ma la cosa sembra non funzionare. Un errore?

Comando:	FILENOTE
Formato:	FILENOTE [FILE] <nome del file> COMMENT <commento>
Sintassi:	FILENOTE "FILE/A,COMMENT/K"
Scopo:	Associa ad un file un commento
Specifiche:	La lunghezza massima del commento è di 79 caratteri. Il commento va racchiuso tra virgolette se contiene spazi od altri caratteri speciali dell'AmigaDOS, come il punto e virgola. Il commento può essere visualizzato con il comando LIST.

Nota: Attenzione: le operazioni riportate nel manuale dell'AmigaDOS 1.3 sono sbagliate. Gli operatori <<e>> vanno sostituiti con <e> come riportato nella tabella. Inoltre gli operatori **eqv**, **mod** e **xor** non funzionano. Si tratta quindi di un comando poco affidabile. Tra l'altro, la presenza di spazi bianchi nell'espressione a volte *confonde* il comando.

TopOffset + IntuiText.TopEdge

e come posizione orizzontale

LeftOffset + IntuiText.LeftEdge

Le strutture **IntuiText** possono inoltre essere concatenate fra loro, permettendo di visualizzarle tutte in una volta con una sola chiamata alla **PrintIText()**.

Ogni struttura della *catena* ovviamente, non solo può rappresentare un differente testo, ma questi può venire visualizzato con attributi differenti, sia per quello che riguarda i colori e la posizione nell'elemento contenitore, sia per quello che riguarda il formato, lo stile e le dimensioni.

Alla **PrintIText()** va inoltre ad aggiungersi un'altra funzione molto utile, sempre riportata in figura 4, già incontrata nella stesura del programma scheletro, e che serve a fornire la lunghezza in pixel di un testo associato ad una struttura **IntuiText**. Questa funzione, chiamata **IntuiTextLength()** è importantissima, dato che non dobbiamo dimenticarci che qui stiamo parlando di testi quali *elementi grafici*, e quindi, non solo da integrare in un discorso di immagini, le quali sono convenzionalmente basate sul pixel come unità di misura, ma le cui reali dimensioni dipendono in effetti dagli attributi definiti nella struttura **TextAttr** referenziata dalla **IntuiText**.

Per quello che riguarda quest'ultima, ne parleremo più diffusamente quando tratteremo dei vari formati *[font]*, in una delle prossime puntate. Per il momento, in figura 2, ho riportato il contenuto della struttura ed una breve descrizione dei vari campi.

L'esercizio

Sono ormai diversi mesi che, tra una cosa e l'altra, non ho più proposto un esercizio ai miei pochi ma fedeli lettori. Ed allora, per la gioia di quei programmatori che si divertono a passare le notti a compilare sull'Amiga dopo una dura giornata di lavoro o di studio, ecco un esercizio semplice semplice per la prossima volta.

Si tratta di scrivere un programmino che apre una finestra sullo schermo del WorkBench e vi disegna in modo casuale una ventina di rettangoli colorati pieni e parzialmente sovrapposti, in modo da formare qualcosa di simile ad un quadro futurista. Lo scopo è quello di generare una immagine a vari colori che faccia da sfondo al nostro testo ma di cui il programma non conosca con precisione le caratteristiche. A questo punto stampate nella finestra una frase, quella che volete, in un certo numero di combinazioni di colori e modi grafici, e vedete un po' quale è la combinazione di modi grafici che garantisce la migliore visibilità indipendentemente dallo sfondo e dai colori utilizzati per il testo. La risposta la sapete già, ma toccare con mano non fa certo male...

Conclusione

Nel prossimo numero vedremo le altre due strutture presentate nell'introduzione di questa puntata, quelle relative cioè agli elementi grafici.

Casella Postale

Una volta che una lettera mi arriva in tempi decenti (spedita il 22 maggio e ricevuta il 4 giugno), mi capita l'estate di mezzo che allunga di un mese i normali tempi di consegna dell'articolo, per cui questo articolo che sto scrivendo tra una partita e l'altra dei Mondiali, non uscirà che in autunno... Non disperate, comunque, e continuate a scrivere. Come sempre cercherò di rispondere a tutti.

_main e _tinymain

Egr. Dott. de Judicibus, seguo con interesse la Sua rubrica dedicata alla programmazione di Amiga in linguaggio C, della quale apprezzo soprattutto l'elevato valore didattico degli esempi di codice proposti.

Per i miei programmi utilizzo il compilatore Lattice (vers. 5.02 aggiornata alla vers. 5.04).

Recentemente mi sono imbattuto in una imprecisione nella documentazione, che ritengo utile segnalare agli altri lettori perché la sua individuazione mi ha fatto perdere del tempo.

A pag. L264 del Vol. 2 vi è la descrizione delle funzioni di libreria _main e _tinymain. Il modulo di startup (generalmente c.o.), prima di cedere il controllo alla funzione main definita dal programmatore, richiama una di esse. _main interpreta la riga di comando con cui viene lanciato il programma, rende accessibili gli argomenti tramite argc e argv, ed apre i file di I/O standard stdin, stdout e stderr.

_tinymain, invece, si limita ad interpretare gli argomenti della linea di comando, evitando di aprire i file di I/O. Se non si impiegano questi file, si può ottenere un programma leggermente più compatto facendo in modo che il linker includa _tinymain al posto di _main nel modulo eseguibile.

Nella citata pagina del manuale si afferma che ciò è possibile specificando

DEFINE _main=_tinymain

sulla linea di comando di blink. In realtà l'opzione corretta è

DEFINE _ _main=_ _tinymain

Infatti, mentre il compilatore aggiunge automaticamente il carattere "_" (codice ASCII 95) come prefisso a tutti gli identificatori, il linker non lo fa e, se si segue l'indicazione del manuale, non è in grado di rintracciare la funzione _ _tinymain nel file di libreria lc.lib.

Cordiali saluti

Paolo Amoroso - Milano

Non c'è molto da aggiungere. Quanto riportato dal sig. Amoroso è corretto. Blink non aggiunge il carattere di sottolineatura ai simboli specificati con l'opzione **DEFINE**. Se infatti, ad esempio, chiamiamo la procedura principale di un programma **_main** invece che **main**, risparmiamo un bel po' di codice specialmente se non è necessario interpretare la riga di comando con cui viene lanciato il programma, se cioè non sono previsti parametri in ingresso. In questo modo, tuttavia, i file di I/O standard non sono aperti, e non è possibile usare funzioni come **printf()**. Questa limitazione può essere aggirata con un trucco. Basta infatti dichiarare una variabile semplice, che verrà interpretata dal compilatore come una variabile di tipo **external int**, assegnare a tale variabile il risultato della chiamata alla funzione AmigaDOS **Output()**, e quindi ridefinire nel comando di assemblaggio *[link]* **stdout** uguale alla variabile in questione, nel modo indicato nel riquadro di tabella A. ▼

Tabella A

```
myout;                  /* Viene interpretata come "external int myout" */
...
_main()
{
  ...
  myout = Output();
  ...
  printf("Stampiamo questa!\n");  /* Fa riferimento implicito a stdout */
  ...
}
```

Nel comando di assemblaggio va quindi specificata la seguente opzione:

```
DEFINE _stdout = _myout
```

proprio in quanto **BLINK** non aggiunge automaticamente il carattere di sottolineatura ai vari simboli.

Desidero ringraziare il sig. Amoroso per la sua lettera, che interpreta nel modo migliore lo scopo di questa rubrica, che è appunto quello di condividere con altri soluzioni ed idee personali, contribuendo così a sviluppare anche in Italia una cultura informatica di livello elevato. MC

NEWEL s.r.l.

FINALMENTE E' ARRIVATO

VIDEON

per IBM PC COMPATIBILI

PREZZO L. 699.000

VIDEON digitalizzatore d'immagine, consente ad un costo estremamente contenuto di catturare immagini a colori che possono essere visualizzate su standard VGA.
Funzionante su computer della classe AT permette di digitalizzare immagini in risoluzioni di:

- 320x200 - 640x400 - 640x480 - in 256 colori
- 1034x768 in 16 colori

Il tempo necessario alla cattura dell'immagine è di 5 secondi B/W e di 50 secondi in 256 colori.
Le immagini catturate col **VIDEON** possono essere elaborate con i più diffusi programmi: PC PAINT - PAGE MAKER - VENTURA - GEM ecc.
Le immagini possono essere salvate nei seguenti modi: P.CX - GIF - PIC

Caratteristiche Tecniche:

Il **VIDEON** è provvisto di un BY PASS, per poter visualizzare l'immagine standard e regolarne luminosità, colore, contrasto, sullo stesso monitor. Funziona con configurazione a 640K di base, (consigliamo 2Mb) livelli di grigio B/W 64, colori 256.
Elaborazione 24 bit, 8 per ogni componente RGB.
Il **VIDEON** funziona in standard parallela.
Tecniche di filtraggio e Dithering fanno si che si ottengano immagini ad altissima qualità.
Immagini che sfruttando tecniche di Dithering avranno apparentemente 200.000 colori.

NEWEL s.r.l. computers ed accessori
20155 MILANO - Via Mac Mahon, 75 - Tel. neg. 02/323492-33000036 - Fax 02/33000035