Programmare in C su Amiga

di Dario de Judicibus

Prima puntata

Iniziamo con questo articolo una nuova serie dedicata alla programmazione in C su Amiga.
Tali articoli si prefiggono come scopo quello di fornire al lettore una visione quanto più completa ed esauriente delle possibilità che il linguaggio C offre su una macchina come l'Amiga. Conseguentemente, una conoscenza anche elementare del C e della programmazione strutturata è prerequisito fondamentale alla comprensione di tali articoli. È opportuno inoltre, che il lettore abbia anche una certa familiarità con l'Amiga DOS tramite interfaccia CLI e con le procedure di compilazione e linkedit del C

Gli esempi riportati nei listati *[listing]* sono scritti seguendo le convenzioni del **Lattice C**, ma potranno comunque essere utilizzati con minime modifiche anche da altri compilatori, grazie alla elevata portabilità di tale linguaggio.

La terminologia utilizzata sarà, nei limiti del possibile, completamente in italiano, per facilitarne la comprensione anche a chi abbia poca dimestichezza con la lingua inglese; tuttavia, dato che molti manuali Amiga non sono stati ancora tradotti in italiano e che comunque ci può essere chi ha parte della propria manualistica in lingua originale, ogni qualvolta comparirà un termine specifico ne verrà data anche la traduzione in inglese. Viceversa, essendo il C un sottoinsieme dell'inglese, e non potendo ovviamente tradurre anche i termini propri del linguaggio stesso *[keyword]*, i listati riportati seguiranno la convenzione inglese anche per le variabili e le funzioni definite dal programmatore. Questo servirà principalmente ad evitare la confusione che potrebbe scaturire dallo scegliere una convenzione bilingue. In ogni caso i commenti saranno sempre in italiano.

Ogni articolo sarà diviso in due o tre parti. La prima parte introdurrà l'argomento che verrà trattato nell'articolo stesso e definirà gli obiettivi che si intende far raggiungere al lettore. La seconda parte costituirà il corpo vero e proprio dell'articolo. La terza parte, non sempre presente, proporrà al lettore un semplice esercizio per verificare se gli obiettivi che erano stati specificati all'inizio dell'articolo sono stati raggiunti. Nella puntata successiva sarà riportata una possibile soluzione dell'esercizio che servirà al lettore come verifica del livello di conoscenza raggiunto.

È importante tener presente che non è assolutamente detto che il codice prodotto dal lettore e quello proposto dall'autore siano identici. È risaputo che in programmazione esistono infiniti modi di ottenere le stesse cose. Di fatto, la migliore verifica della bontà dei propri programmi si ha solo quando si cerca di compilarli e di farli girare *[run]* sul computer stesso. Tuttavia è sempre possibile ricavare dal listato proposto come soluzione all'esercizio qualche buona idea per migliorare il proprio codice *[code]*.

Introduzione

In questo primo articolo introdurremo l'Architettura Software di Sistema dell'Amiga, definiremo i concetti base e la terminologia necessaria per quando ci inoltreremo più a fondo nella programmazione in C dell'Amiga.

Gli obiettivi che ci riproponiamo di raggiungere sono i seguenti:

- essere in grado di aprire una libreria in fase di esecuzione;
- sapere come verificare il successo dell'operazione effettuata;
- chiudere la libreria dopo averla utilizzata.

Il Software di Sistema

Il Software di Sistema *[kernel]* è quell'insieme di moduli che permettono di utilizzare l'hardware di una macchina. Sebbene sia sempre possibile controllare direttamente l'hardware via software, ciò richiede una conoscenza più che buona della struttura interna della macchina.

L'Amiga mette tuttavia a disposizione dei programmatori tutta una serie di moduli distribuiti su quattro differenti livelli. Ogni livello utilizza i servizi di quello sottostante, come mostrato in figura 1. Il livello più basso è quello immediatamente al di sopra dell'hardware, mentre quello più alto è quello che interfaccia direttamente l'utilizzatore del Sistema, od utente *[user]*.

Parte dei moduli del Software di Sistema risiedono permanentemente in una area di memoria protetta della macchina, quella cosiddetta del **KickStart**. Nell'Amiga 1000 il KickStart («Calcio di avvio») viene caricato da dischetto su

RAM *[cold boot]*, cioè su di un'area di memoria che si cancella allo spegnimento della macchina.

Tale area tuttavia viene ad essere protetta in modo tale da poter far ripartire «a caldo» *[warm boot]* l'Amiga senza doverla ricaricare di nuovo. La partenza a caldo si effettua premendo contemporaneamente i tasti
CTRL + Left AMIGA + Right AMIGA.

Nell'Amiga 500 e 2000, il KickStart è già presente sulla macchina in ROM, cioè in un componente *[chip]* hardware montato sulla scheda del computer.

La restante parte del Kernel che non si trova nel KickStart, viene caricata quando serve dal dischetto di Sistema, da quello cioè che contiene il **Work-Bench**.

Vediamo adesso in maggior dettaglio la struttura a livelli del Kernel Amiga. Faremo sempre riferimento alla figura 1. Prima però, è bene definire due termini: *task* e *processo [process]*:

task: è un insieme di istruzioni con un compito specifico (in inglese task significa «lavoro»), che, durante l'esecuzione, ha il controllo in esclusiva dei registri hardware della macchina e può utilizzare tutte le risorse del sistema condividendole con gli altri task caricati. Dato che solo un'istruzione alla volta può essere eseguita dal microprocessore, quando un task è in esecuzione, tutti gli altri sono «addormentati» in attesa di un segnale specifico *[interrupt]* che li svegli. Quando un task viene svegliato, lo stato del task precedente viene salvato in un'area di memoria, mentre quello del nuovo task viene rigenerato come era al momento dell'interruzione. Un interrupt può essere generato sia da un altro task, sia da una periferica esterna o dal meccanismo di controllo del multitasking.

Processo: un processo è un superinsieme del task. Esso è formato da un task più altre strutture di dati utilizzate dall'AmigaDOS.

La conoscenza di tali termini è importante in un sistema multitasking quale è l'Amiga. Data tuttavia la complessità di tale argomento, rimandiamo agli ultimi articoli la spiegazione su come scrivere programmi formati da più task in comunicazione fra di loro. Tornando quindi alla struttura del Software di Sistema, partiamo dal livello più basso, quello cioè a più stretto contatto con l'hardware. Esso è formato di sei parti *[set]*:

EXEC: è l'insieme dei moduli che controllano direttamente il processore principale dell'Amiga, cioè il Motorola 68000. Tali moduli sono responsabili della gestione *multitasking* del 68000. Essi inoltre si occupano di allocare la memoria che serve ai vari *tasks* ed a gestire gli *interrupts* che sono generati dai chip speciali (Agnus, Denise and Portia) e dal software applicativo. Ve-

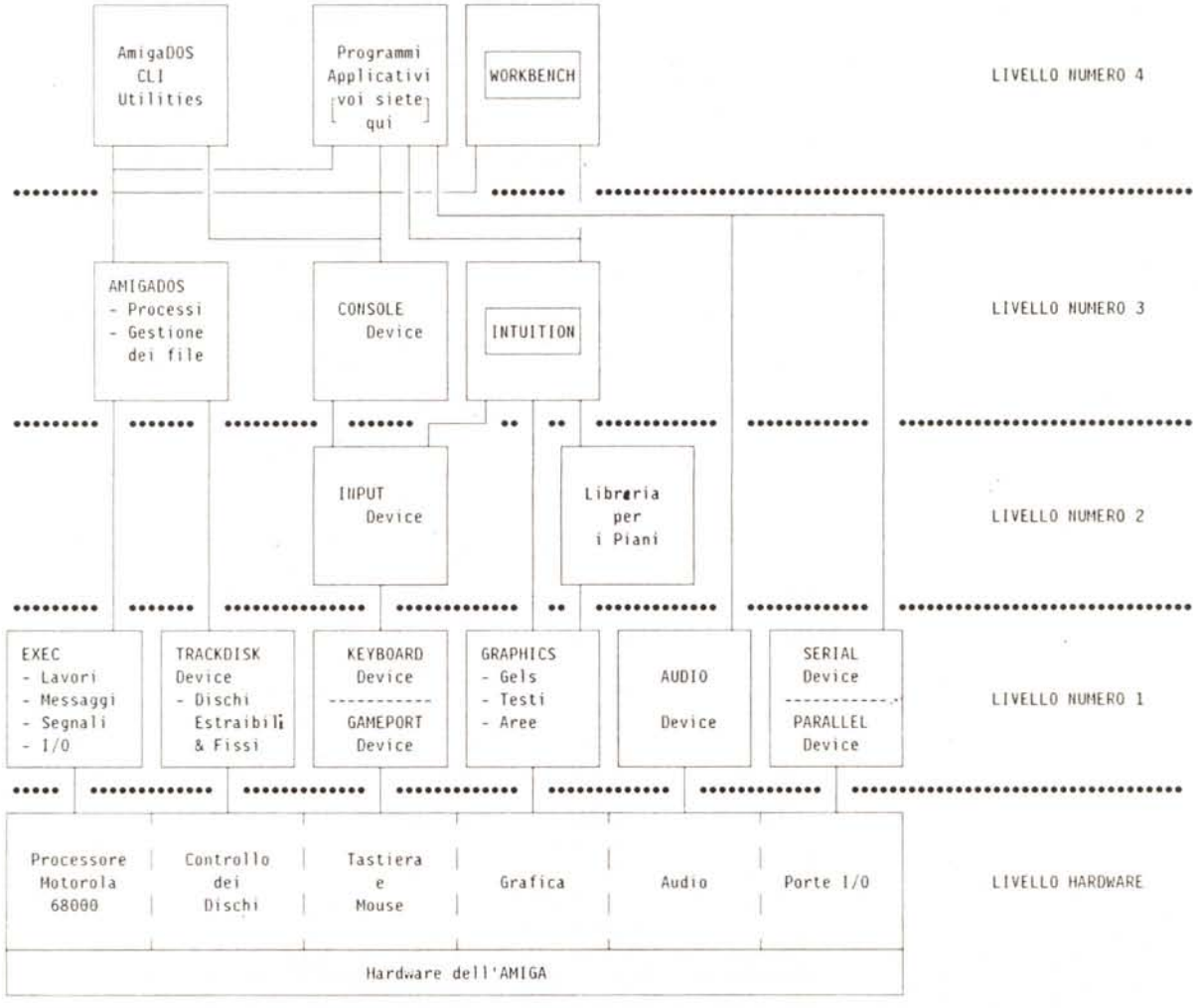

Figura 1 - Il Software di Sistema Amiga.

```
struct NewWindow
{
  SHORT LeftEdge, TopEdge;   /* Angolo in alto a destra della finestra  */
  SHORT Width, Height;       /* Larghezza ed altezza della finestra     */
  UBYTE DetailPen, BlockPen; /* Colori delle varie parti della finestra */
  ULONG IDCMPFlags;          /* Gestione degli eventi da Intuition      */
  ULONG Flags;               /* Caratteritiche della finestra stessa    */
  struct Gadget *FirstGadget;/* Lista dei gadgets associati             */
  struct Image *CheckMark;   /* Simbolo usato per marcare un menu item  */
  UBYTE *Title;              /* Titolo della finestra                   */
  struct Screen *Screen;     /* Schermo a cui appartiene la finestra    */
  struct BitMap *BitMap;     /* Area usata da finestre SUPER_BITMAP     */
  SHORT MinWidth, MinHeight; /* dimensioni minime ammesse               */
  SHORT MaxWidth, MaxHeight; /* dimensioni massime ammesse              */
  USHORT Type;               /* Tipo di schermo utilizzato              */
};
```

Figura 2 - Struttura NewWindow.

dremo nelle prossime puntate il significato di tali termini e l'utilizzo dei moduli di EXEC.

Trackdisk Device: è l'interfaccia di livello più basso per la gestione dei dischi, sia di quelli fissi che di quelli estraibili.

Keyboard e Gameport Device: sono le interfacce verso le periferiche di ingresso al sistema *[Input Devices]*, cioè:

- la tastiera *[keyboard]*
- il mouse
- il joystick
- eventuale altro hardware che si connetta alle prese del mouse e del joystick *[gameports]*.

Audio Device: interfaccia l'hardware relativo al Sistema Audio dell'Amiga.

Device Seriale e Parallela: gestiscono le due porte seriale e parallela che servono per la connessione di varie periferiche esterne come, ad esempio, stampanti e plotter.

Grafica: l'insieme dei moduli che formano il *set* grafico gestisce l'hardware per la grafica e permette di compiere varie funzioni grafiche come, ad esempio, disegnare linee, riempire aree, selezionare colori e sfondi *[pattern]*, gestire oggetti grafici.

Incontreremo spesso il termine *device*, purtroppo intraducibile in italiano. Esse sono molto importanti nell'architettura Amiga. Vedremo in seguito di cosa si tratta quando parleremo di EXEC in una delle prossime puntate.

Il secondo livello è costituito da due parti:

Input Device: è un task indipendente del sistema, che intercetta le informazioni in ingresso *[input events]* dalla keyboard e dalla Gameport Device e le unisce *[merge]* in modo da formare un unico flusso di dati *[input data stream]* che può essere utilizzato da alcuni componenti del livello superiore.

Layers Library: è un insieme di procedure che permettono di creare diversi piani *[layers]* grafici che si possono sovrapporre e che quindi formano la base di un sistema a finestre *[window environment]* quale siamo abituati ad utilizzare sull'Amiga.

Il terzo livello è formato da tre parti, estremamente importanti:

AmigaDOS: è il Sistema Operativo vero e proprio *[Disk Operating System]*. È un sistema multiprocesso che utilizza EXEC per permettere ai vari processi di condividere *[share]* le risorse del sistema, compreso il processore centrale, condiviso tra vari task. Grazie ad esso è possibile gestire i dati sotto forma di *file* e *directory*. AmigaDOS mette inoltre a disposizione dell'utente vari programmi di utilità *[utilities]* come ad esempio quelli che impostano la data e l'ora di sistema, listano a terminale il contenuto di un file ASCII o forniscono informazioni su un dischetto inserito nell'apposita unità *[driver]*.

Intuition: è l'interfaccia multischermo e multifinestra dell'Amiga. Essa è composta da varie funzioni facenti parte di un'unica libreria, che permettono al programmatore di definire schermi *[screen]*, finestre *[window]*, menu *[menu]*, solleciti *[requester]*, vari strumenti di immissione dati *[gadget]* e via dicendo, come vedremo in una delle prossime puntate. Essa inoltre seleziona e rende disponibili al programmatore solo quegli eventi che interessano fra quelli che vengono forniti dall'Input Device.

Console Device: viene associata ad una finestra di Intuition per simulare il comportamento «classico» di un terminale.

Per finire, abbiamo il quarto livello, quello cioè a stretto contatto con l'utente. Questo livello non è suddivisibile in parti, in quanto di esso fanno parte tutti i programmi applicativi che utilizzano il Software di Sistema vero e proprio. Quest'ultimo, infatti, è formato più propriamente solo dai livelli uno, due e tre, anche se due importantissimi e più o meno conosciuti programmi forniti con il Sistema Operativo appartengono di fatto al livello più alto, il quarto appunto:

CLI: è un programma applicativo che permette di far girare altre applicazioni e di lanciare comandi dell'AmigaDOS tramite una finestra gestita dalla Console Device.

WorkBench: come il CLI, esso permette di lanciare altri programmi, utilizzando tuttavia un'interfaccia più amichevole, anche se in certi casi meno flessibile di quella CLI. Il WorkBench è quella che si chiama una interfaccia *object oriented*, dato che l'utente interagisce con essa lavorando su simulazioni di oggetti reali che vengono manipolate tramite icone *[icon]*, cioè immagini grafiche che rappresentano vari oggetti di uso comune e che si comportano come tali: cassetti *[drawer]*, dischetti *[disk]*, progetti *[project]* e via dicendo.

A questo livello apparterranno quindi anche tutti i programmi che scriveremo e la maggior parte di quelli disponibili in commercio o come *Software di Pubblico Dominio [PD SW]*.

Il C e l'Amiga

Molti sono i linguaggi disponibili per Amiga, ma, se escludiamo l'Assembler, peraltro non alla portata di tutti, quello che meglio si sposa con l'Amiga è il C. D'altra parte tutte le strutture e le informazioni disponibili sui manuali ufficiali della Commodore sono attualmente riportate in questi due linguaggi. Di fatto, il Sistema Operativo dell'Amiga è stato scritto in gran parte in BCPL, un «antenato» del moderno C.

Per chi voglia approfondire quanto diremo in questa serie di articoli, si raccomanda la lettura dei *ROM Kernel Manuals (RKMs)*, i quali riportano inoltre anche buona parte delle strutture contenute negli Include Files che utilizzeremo nel nostro codice. Non si tratta tuttavia di una lettura facile, e sarebbe bene affrontarla dopo aver letto attentamente questi articoli o qualsiasi altro libro di divulgazione sulla programmazione su Amiga. Ce ne sono ormai diversi sul mercato. Non so se sia già arrivato in Italia l'aggiornamento dei RKMs per l'AmigaDOS 1.2, comunque anche quelli per la versione 1.1 possono fornire molte utili informazioni al programmatore esperto.

Vediamo ora di definire alcuni concetti base che ci serviranno in seguito quando incominceremo a parlare più dettagliatamente di programmazione. Non spaventatevi se non vi sarà subito tutto chiaro fin dall'inizio. Man mano che andremo sempre più a fondo nel nostro piccolo corso di programmazione su Amiga le cose diventeranno sempre più semplici e di facile apprendimento.

Strutture: le strutture *[structure]* rappresentano in Amiga uno degli strumenti fondamentali per trasferire dati tra differenti funzioni. Esse possono essere anche molto complesse e permettono al programmatore di ottenere una quantità impressionante di informazioni impostate da alcuni dei componenti del Sofware di Sistema quali Intuition od EXEC. Una struttura «classica» e che incontreremo di frequente è quella che serve a definire una finestra in uno schermo: NewWindow (vedi figura 2).

Alcune considerazioni sull'esempio riportato:

1. Innanzitutto osserviamo che una struttura può contenere altre strutture al suo interno, o comunque puntatori ad altre strutture. Vedremo che questa caratteristica è spesso utilizzata nelle strutture che useremo in seguito.

2. In secondo luogo è importante ricordare che è necessario sempre rispettare la convenzione mista per i nomi delle variabili, delle funzioni ed in generale per tutti gli elementi della sintassi C. I due nomi MyStruct ed mystruct rappresentano quindi strutture differenti. Mettere un carattere in maiuscolo piuttosto che minuscolo o viceversa, può quindi portare ad un errore durante la compilazione o peggio ancora in esecuzione. Fate molta attenzione, quindi. Sempre riguardo le convenzioni utilizzate nello scrivere codice C, è buona norma scrivere in *maiuscolo* tutti i nomi definiti tramite *#define* e *typedef*.

3. Infine è bene utilizzare sempre i tipi predefiniti riportati in figura 3 per garantire la massima portabilità del proprio codice tra più sistemi o computer differenti.

```
#define GLOBAL extern
#define IMPORT extern
#define STATIC static
#define REGISTER register
#define VOID      void

typedef long            LONG;
typedef unsigned long   ULONG;
typedef unsigned long   LONGBITS;
typedef short           WORD;
typedef unsigned short  UWORD;
typedef unsigned short  WORDBITS;
typedef char            BYTE;
typedef unsigned char   UBYTE;
typedef unsigned char   BYTEBITS;
typedef unsigned char   *STRPTR;
typedef STRPTR          *APTR;

typedef short           SHORT;
typedef unsigned short  USHORT;

typedef float           FLOAT;
typedef double          DOUBLE;
typedef short           COUNT;
typedef unsigned short  UCOUNT;
typedef short           BOOL;
typedef unsigned char   TEXT;

#define TRUE      1
#define FALSE     0
#define NULL      0

#define BYTEMASK  0xFF
```

Figura 3 - Tipi predefiniti in exec/types.h.

Libreria (.library)	Puntatore Base	Contenuto:
clist	ClistBase	gestione di stringhe di caratteri
diskfont	DiskfontBase	gestione dei tipi (font) di caratteri
exec	ExecBase	procedure EXEC
dos	DosBase	procedure DOS
graphics	GfxBase	funzioni grafiche
icon	IconBase	gestione degli oggetti del WorkBench
intuition	IntuitionBase	funzioni dell'interfaccia Intuition
layers	LayersBase	gestione dei layer grafici
mathffp	MathBase	matematica base
mathtrans	MathTransBase	matematica trascendentale
mathieeedoubbas	MathIeeeDoubBasBase	matematica in doppia precisione IEEE
timer	TimerBase	aritmetica per il timer
translator	TranslatorBase	traduzione fonemi

Figura 4 - Librerie di Sistema e rispettivi puntatori Base.

Librerie: d'ora in poi, quando parleremo di librerie, non intenderemo tanto quelle librerie che si accedono in fase di compilazione e che permettono di utilizzare le funzioni e le macro fornite con il compilatore quali ad esempio la classica printf() oppure isspace(), strcat() e simili, quanto piuttosto le cosiddette librerie di esecuzione *[run-time libraries]*. Queste sono un insieme di funzioni correlate logicamente dato che ogni libreria si occupa di fornire servizi relativi ad un ben specifico aspetto del sistema. Ad esempio, la intution.library contiene tutte quelle funzioni che permettono di gestire l'interfaccia a finestre tipica dell'Amiga. Viceversa è necessario aprire la libreria mathfpp.library se si vogliono utilizzare le procedure per la matematica a virgola mobile (Fast Floating Point Routines).

Una libreria consiste in un file che contiene una sorta di indice *[Jump Table]* che specifica la posizione nella libreria di ogni singola routine, e dalla successione vera e propria delle procedure.

Parte delle librerie si trova nell'area protetta del KickStart, parte si trova nella *directory* **Libs**: del dischetto da cui si è fatta la partenza *[bootstrap]*. Per potere utilizzare una qualunque routine di una libreria, bisogna quindi innanzitutto ricavare la posizione in cui è stata (KickStart) o sarà (Libs:) caricata la libreria in questione. Dato che tale posizione è scelta volta per volta dal sistema, è necessario chiedere al sistema stesso il puntatore alla libreria che si intende aprire. Per farlo si usa la routine OpenLibrary() che si trova nella exec.library. Questa infatti, insieme alla dos.library, viene aperta automaticamente dal codice di inizializzazione *[startup]* a cui bisogna collegarsi nella fase di linkedit del programma. Si assume che il lettore abbia familiarità con i processi di compilazione e link di un programma C. Eventualmente fate riferimento al manuale del vostro compilatore preferito.

Il puntatore che viene fornito dalla OpenLibrary si chiama **Base**. In figura 4 è riportata la lista delle librerie di sistema ed i nomi dei puntatori Base. Tali nomi sono fissi e non vanno assolutamente cambiati!

Una volta aperta una libreria, il puntatore Base non cambia fino alla chiusura della stessa, tramite CloseLibrary. Se per un qualunque motivo il sistema non riesce ad aprire la libreria richiesta, OpenLibrary ritorna il valore NULL, cioè zero. Quando si utilizzano delle librerie «run-time», è bene seguire le seguenti precauzioni:

1. Includere sempre exec/types.h prima di ogni altro comando #include;
2. definire la OpenLibrary come EXTERNAL;
3. verificare sempre che il puntatore Base non sia nullo;
4. non usare mai una routine di libreria se prima non si è aperta la stessa o la Base è nulla;
5. chiudere sempre le librerie aperte prima di uscire dal programma, sia alla fine che in caso di uscita forzata *[abort]*.

In figura 5 è riportato un esempio di apertura e chiusura «pulita» di una libreria.

Il secondo parametro della OpenLibrary serve a specificare la versione del Software di Sistema che si vuole utilizzare. Ad esempio, per l'AmigaDOS 1.1 il numero della versione è 31. Se non interessa assicurarsi di stare utilizzando routine di una versione piuttosto che di

```
/*
 *  Come aprire e chiudere in modo corretto una libreria (intuition)
 */
#include "exec/types.h"
#include "intuition/intuition.h"
#include "intuition/intuitionbase.h"
extern struct Library *OpenLibrary();
LONG IntuitionBase *IntuitionBase

main()
{
  /*
   *  Apri la libreria. Se qualcosa va male, esci.
   */
  if ((IntuitionBase = (struct IntuitionBase*)
        OpenLibrary("intuition.library",0L)) == NULL)
  {
     printf("Non posso aprire la libreria Intuition\n");
     Exit(0L);
  }

  /*
   *  Corpo del programma: utilizza le procedure della libreria aperta
   */
   .
   .
   .

  /*
   *  Chiudi la libreria prima di uscire definitivamente
   */
  CloseLibrary(IntuitionBase);
}
```

Figura 5 - Apertura e chiusura di una libreria.

```
if (LibNBase == NULL)
{
  CloseLibrary(Lib1Base);
  CloseLibrary(Lib2Base);
  CloseLibrary(Lib3Base);
  .
  .
  .
  Exit(0L);
}
```

Figura 6

un'altra, basta impostare tale valore a zero.

L'esercizio

Quando si devono aprire più librerie una dopo l'altra, è necessario, qualora una delle richieste fallisca, chiudere anche tutte quelle precedenti. Questo vuol dire scrivere dopo ogni controllo una sequenza di CloseLibrary come indicato in figura 6.

Questo può essere abbastanza scomodo e poco flessibile. Infatti, nel caso sia necessario modificare il programma aggiungendo un'altra libreria, occorre modificare tutto il codice di controllo delle aperture successive.

Per ovviare a ciò ci sono molte tecniche. Nella puntata successiva ve ne presenteremo una. Per il momento provate voi ad inventarvi un trucco che eviti tale appesantimento del codice e semplifichi le modifiche future. Magari risulterà migliore del nostro! Parleremo inoltre del terzo livello, e più precisamente di AmigaDOS.

Buon lavoro!

MC