

Ottobre 1997

**PROGRAMMAZIONE  
IN AMBIENTE  
MICROSOFT WINDOWS**

**Visual C++**

Introduzione all'ambiente Windows  
e strumenti di programmazione

*Ing. Stefano Riccio  
stefaric@tiscalinet.it*

# BIBLIOGRAFIA

Per quanto riguarda la programmazione in 'C' utilizzando SDK:

- **Manuali Visual C++**
- **Manuali Microsoft SDK (Software Development Kit)**
- **Programmare in Windows. Gestione dell'input e uso delle risorse e Programmare in Windows. GDI, DDE e collegamenti**  
*Petzold* ed. Mondadori Informatica

Per quanto riguarda la programmazione in 'C++' utilizzando MFC:

- **Manuali Visual C++**
- **Microsoft Visual C++, Programmare in MFC e Win 32**  
ed. Mondadori Informatica
- **Il manuale Visual C++**  
*David J. Kruglinski* ed. McGraw Hill

& Altri testi utili:

- J. Conger, Microsoft Foundation Class Primer, Wait Group Press, 1993
- B. Stroustrup, Il linguaggio C++, Addison Wesley, 1993
- S. Holzner, Programmazione in Visual C++, Jackson libri, 1993
- C. Simonelli - C. Munisso, Dal C a Windows passando per il C++, 1995

## Cos'è Windows

Prima di tutto cerchiamo di capire cos'è Windows, quali sono le sue caratteristiche e quali vantaggi offre.

Windows non è un Sistema Operativo, anche se svolge molti compiti propri di quest'ultimo. Viene spesso usato il termine "Ambiente Operativo" che mette in risalto la gradevole e semplice interfaccia di Windows. Windows è, comunque, molto più di un ambiente con una interfaccia user-friendly; le sue maggiori qualità sono legate all'indipendenza del codice delle applicazioni dall'hardware sottostante e agli innumerevoli strumenti predefiniti che offre (soprattutto per la grafica).

Un PC è costituito da un'insieme di parti, che sono fondamentalmente dei circuiti: *cpu, memoria, tastiera, schermo, dischi ecc.* La comunicazione tra software e questi circuiti avviene in vario modo: *interrupt* effettuati da un elemento dell'hardware che generano una chiamata di funzione ad un indirizzo in dipendenza del tipo di interrupt, *operazioni di mappatura* a degli indirizzi in memoria (come avviene per lo schermo) oppure utilizzo delle porte di I/O.

Per rendere il software relativamente indipendente dall'hardware ed offrire una programmazione più semplice rispetto al controllo diretto dell'hardware, i PC integrano un primo strato di software, il **BIOS**. Al di sopra del BIOS si colloca il sistema operativo, per esempio il Dos. Quest'ultimo è un sistema che adopera le primitive del BIOS per accedere all'hardware ed offre servizi più sofisticati, in particolare per la gestione dei dischi e dei file.

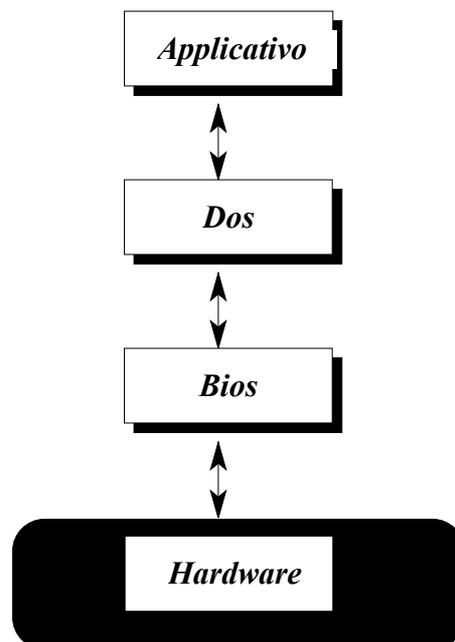


Figura 1 Accesso all'hardware attraverso DOS e BIOS

Un'applicazione sviluppata per un PC sotto Dos accede normalmente all'hardware attraverso il Sistema Operativo stesso (figura 1).

Qui iniziano i problemi, in quanto il Dos è carente nella gestione della tastiera e del video, di conseguenza molti programmatori preferiscono scavalcarlo e accedere all'hardware chiamando direttamente le primitive del BIOS, se non addirittura lo stesso hardware (figura 2).

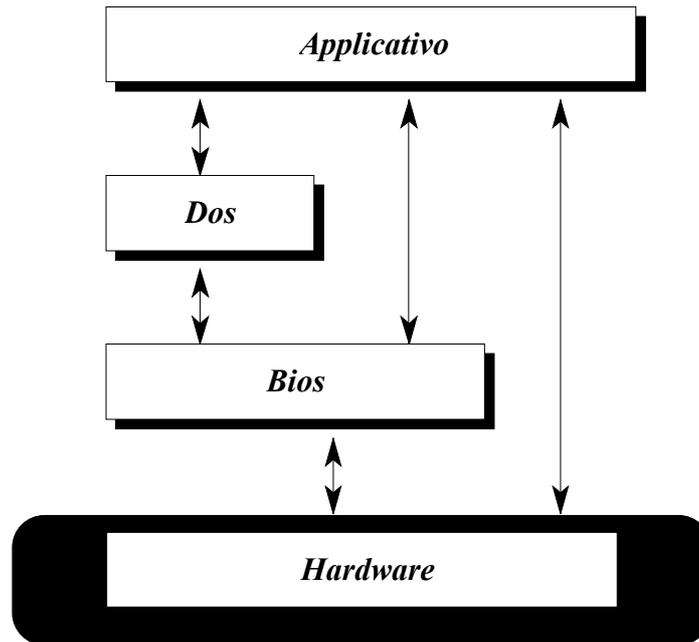


Figura 2 Accesso diretto all'hardware

È non è tutto, ad esempio l'integrazione di un nuovo elemento hardware, non noto al BIOS, è difficoltosa: bisogna aggiornare il BIOS oppure l'applicazione deve accedere direttamente al nuovo elemento.

Windows è stato creato per evitare al programmatore di preoccuparsi del tipo di hardware sul quale gira il proprio applicativo. Tutte le periferiche abituali di un microcomputer sono virtualizzate: *tastiera, mouse, video, stampante, tavoletta grafica, mezzi di comunicazione*, offrendo un'interfaccia standard, perfettamente definita e completa, totalmente indipendente dall'hardware installato.

### Confronto Dos-Windows

Nel paragrafo precedente si sono già messe a fuoco differenze tra Dos e Windows, e in fondo è inevitabile a questo punto non confrontarli. Ecco cosa offre in più Windows rispetto allo standard Dos:

- una interfaccia grafica caratterizzata da finestre, menu, controlli ecc..
- code di input;
- device grafiche indipendenti;
- multitasking.

#### Interfaccia grafica

Uno degli obiettivi principali di Windows è dare accesso agli utenti a tutte le applicazioni nello stesso tempo. Questo è stato realizzato tramite la condivisione dello schermo tra le applicazioni. Windows assegna ad ogni applicazione una parte dello schermo e quindi l'utente ha sottocchio tutte le applicazioni.

Un'applicazione condivide il display usando una finestra, ossia un rettangolo che rappresenta una porzione dello schermo, in realtà una finestra è una combinazione di device, come *menu', scrollbar* ecc.. Quindi a differenza del Dos, dove lo schermo è tutto a disposizione, il programmatore, all'inizio del suo programma, deve sempre crearsi la propria finestra di lavoro.

### **Code di Input**

Una delle principali differenze tra Dos e Windows è come viene ricevuto l'input. In Dos si fa una chiamata a una funzione che restituisce ad esempio un carattere letto dalla tastiera. Windows riceve tutto l'input dalla tastiera, dal mouse, dal timer e lo mette in una *coda messaggi*. Quando l'applicazione deve leggere dell'input, legge il prossimo messaggio nella coda.

Un messaggio è una struttura più complessa dell'input del Dos, ma permette di trattare nello stesso modo tutto l'input indipendentemente da dove esso arriva.

Questo meccanismo permette di virtualizzare la tastiera, Windows contiene le *device drivers* di un certo numero di tipi diversi di tastiera; queste device drivers convertono i caratteri in messaggi standard che Windows riconosce.

### **Device grafiche indipendenti**

In Windows si ha accesso ha un ricco insieme di operazioni su device grafiche, questo significa che l'applicazione può facilmente disegnare cerchi, ellissi nella propria regione. Inoltre Windows virtualizza lo schermo, o meglio tutto l'output, quindi i comandi sono unici per scrivere o disegnare su una CGA come su una VGA, e si usano le stesse funzioni anche per stampare o per usare un plotter.

Anche qui Windows ha bisogno di una device driver per convertire il comando in codice per la periferica, quindi risulta molto semplice (a differenza di come visto con il BIOS) adottare una nuova periferica purché Windows o il costruttore ci fornisca la device driver ad essa associata.

### **Multitasking**

Un discorso a parte merita il *multitasking*, in considerazione del fatto che tutti i testi da me consultati parlano di Windows come un tale sistema. In realtà in Windows il termine Multitasking è inteso come condivisione di risorse; a differenza del Dos, un'applicazione non occupa tutte le risorse, ma le condivide con tutte le altre applicazioni. Esiste multiprogrammazione in quanto più processi possono stare contemporaneamente in memoria, ma i processi non sono certo spezzati in task ed eseguiti in modo concorrente.

In sistemi di multitasking reale mentre un processo effettua operazioni di I/O, la CPU passa ad elaborare un altro task. Io stesso ho provato, in ambiente Windows, ad eseguire programmi che utilizzano molto I/O; quando si cambia applicazione essi non continuano ad elaborare in background come prevede il multitasking reale.

Questo non è più vero in ambiente Windows NT, dove è stato implementato un meccanismo a divisione di tempo (time sharing) tra i processi in memoria.

## **Punto di vista del programmatore**

Vediamo ora di capire cosa cambia per il programmatore abituato a sviluppare programmi sotto Dos quando entra nel mondo Windows. Windows purtroppo presenta un duplice aspetto; se da una parte per l'utente finale risulta molto semplice da utilizzare, potente e di immediato apprendimento, dall'altra risulta (almeno in un primo impatto) confuso, di difficile accesso per il programmatore.

### **Processo di costituzione di un applicativo**

Mettiamo in evidenza la differenza procedurale di costituzione di un applicativo Dos (figura 3) e di un applicativo Windows (figura 4).

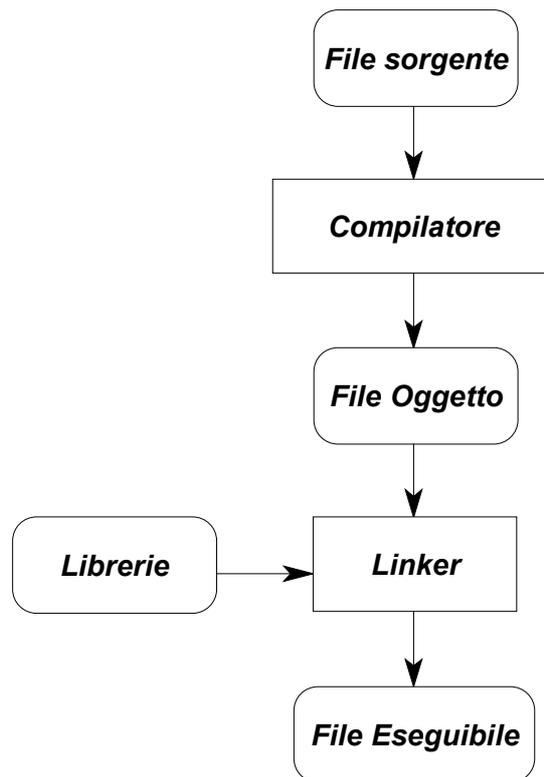


Figura 3 Applicativo sotto DOS

Come si può vedere dalle figure il processo in Windows si complica un po', nasce una attività parallela rappresentata dalla compilazione delle risorse. Il file sorgente di base è il *file delle risorse* (di estensione **RC**), scritto in un linguaggio descrittivo particolare. Esso contiene la descrizione delle risorse utilizzate dall'applicativo, può trattarsi di: *cursori, immagini bitmap, icone, menu ecc..*

Le risorse devono essere compilate con un apposito compilatore (**Resource Compiler**) ottenendo così il file oggetto di estensione **.RES**; quest'ultimo deve essere collegato, tramite il linker, all'eseguibile dal medesimo Resource Compiler.

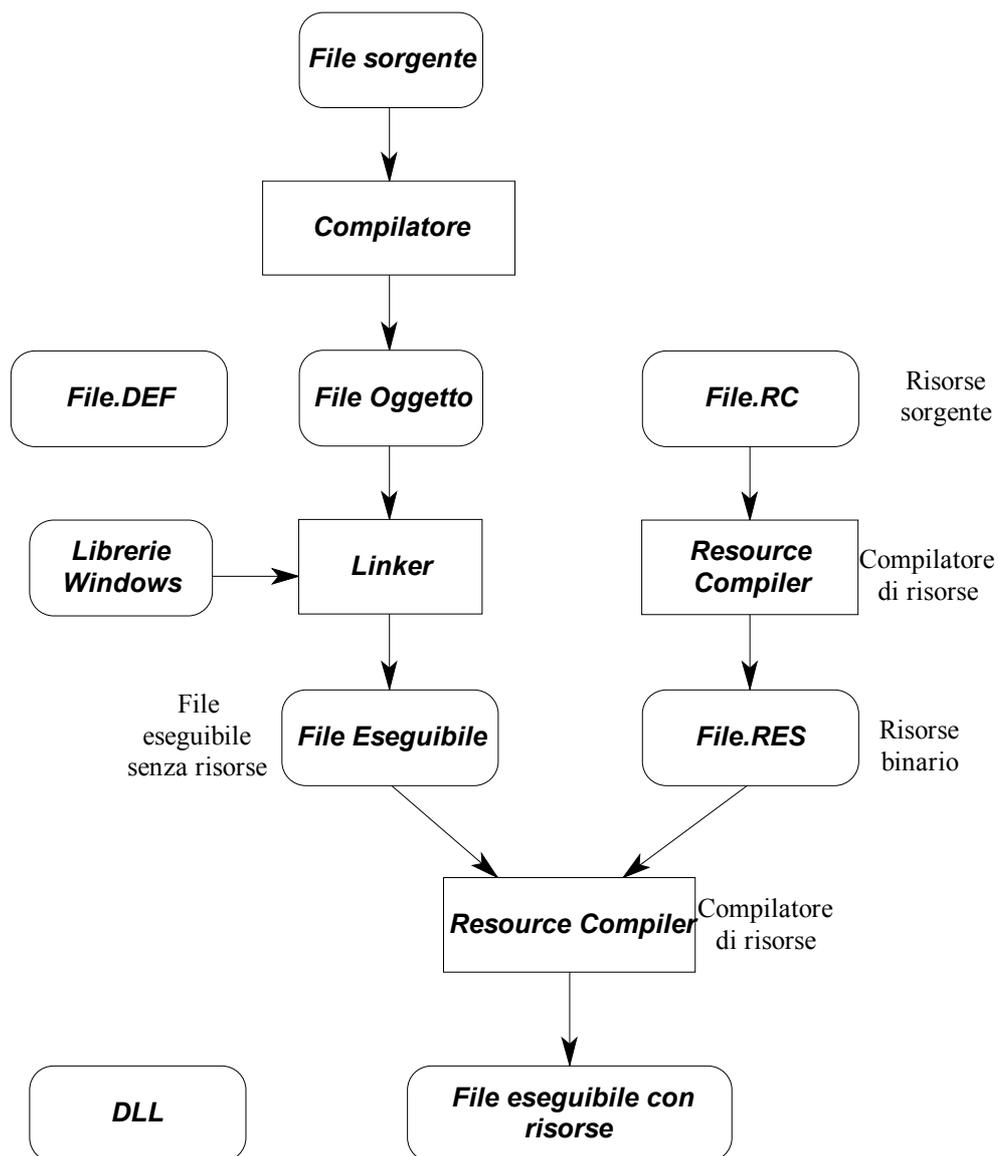


Figura 2.4 Applicativo sotto WINDOWS

Altre due novità sono i files di estensione **.DEF** che devono contenere informazioni riguardo la memoria da assegnare all'applicativo in Windows e le *librerie dinamiche*, ossia librerie di funzioni che vengono collegate in fase di esecuzione.

#### □ **Struttura del programma principale**

Abbiamo parlato dei messaggi e di come viene processato l'input in ambiente Windows, questo comporta per il programmatore il cambiamento della struttura dei propri programmi. Generalmente la struttura di un programma sotto Dos è quella presentata in figura 5.

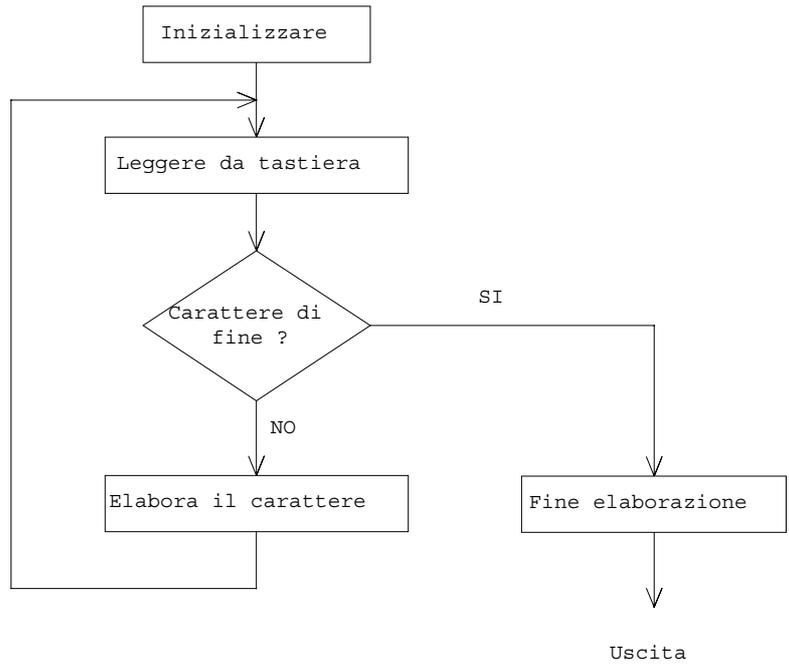


Figura 5 Struttura di un programma DOS

Non differisce molto il corpo principale di una applicazione Windows se non per il tipo di input (figura 6).

Quindi non si leggono più caratteri, ma messaggi che possono provenire da innumerevoli fonti.

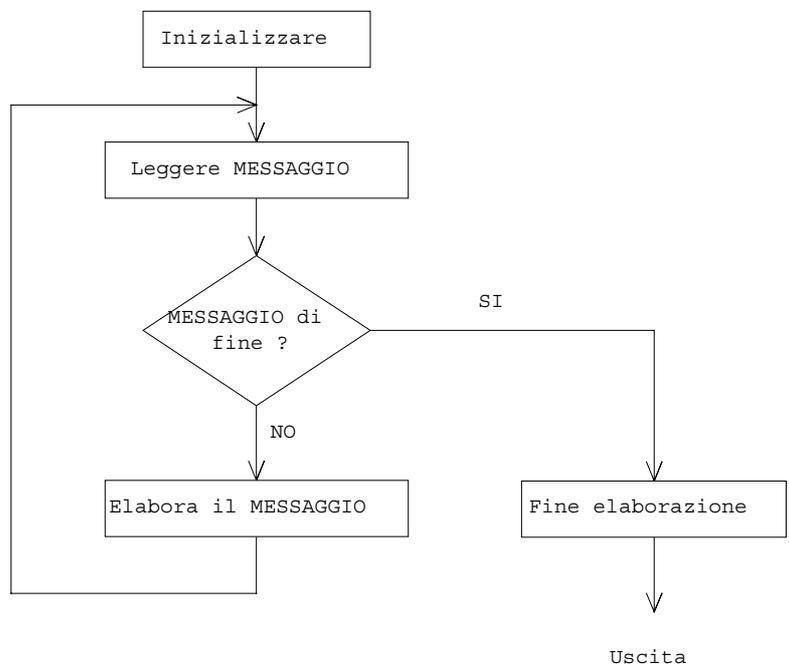


Figura 6 Struttura di un programma WINDOWS

**Message Loop**

Vediamo di approfondire meglio come i messaggi vengono elaborati da Windows e di conseguenza come devono essere trattati dal programmatore. Windows colleziona tutto l'input nella *system queue* (coda unica nel sistema) e trasferisce i messaggi nelle varie *application queues* (una coda per ogni applicazione) (figura 7).

Ogni applicazione deve essere in grado di accettare i messaggi, ma a differenza del Dos, essi non vengono elaborati direttamente, chiamando ad esempio una funzione, ma vengono distribuiti a ogni "Window function".

Qualunque applicazione in Windows è formata da:

- una funzione principale (come il main del c) denominata WinMain;
- un insieme di funzioni, dette window functions, ognuna delle quali è collegata a una finestra del mio applicativo.

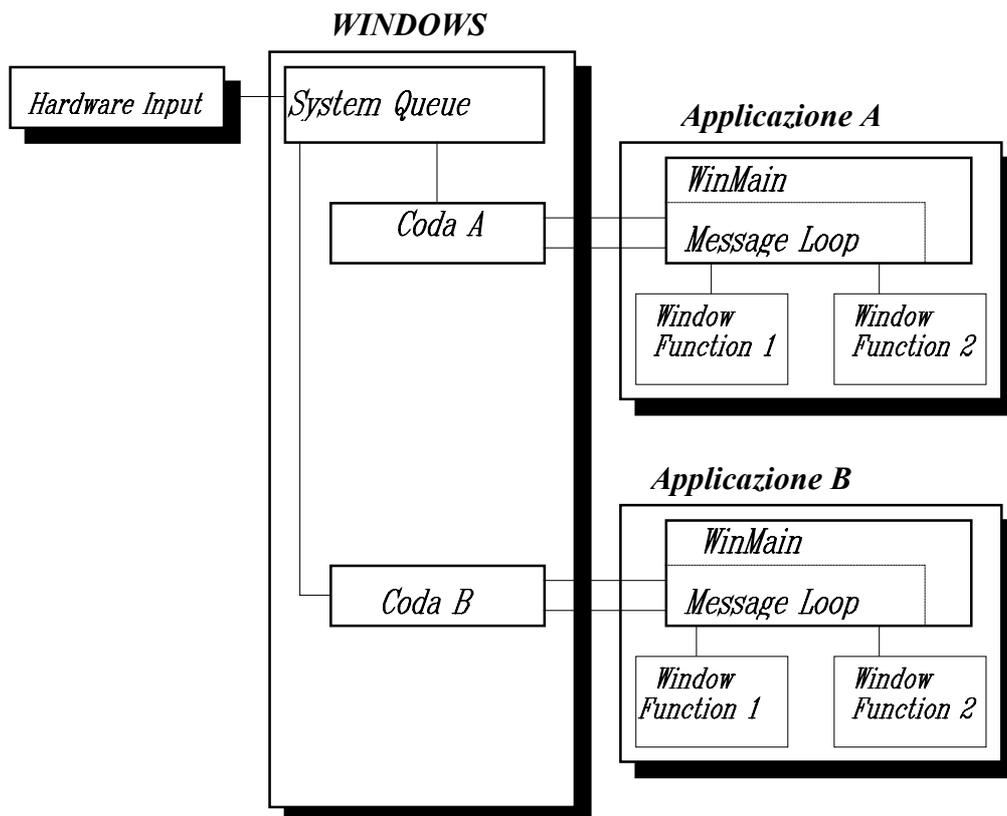


Figura 7 Trattamento dei messaggi in WINDOWS

La WinMain deve quindi leggere tutti i messaggi e distribuirli alle varie window functions che si prenderanno carico di eseguire certe operazioni per ognuno dei possibili messaggi. Per fare questo ogni applicazione ha un "message loop", ossia un ciclo di lettura e distribuzione messaggi.

Il programma, quindi, interagisce continuamente con Windows, non ci sarà mai una chiamata diretta a una funzione associata a una finestra, ogni collegamento tra window functions del mio programma avviene attraverso un messaggio.