Anni fà tutte le librerie disponibili oggi e i sistemi di sviluppo collegati ai sistemi operativi erano sogni. Se si voleva gestire un database mediante gestioni btree l'unica soluzione era quella di studiare l'algoritmo e scriverlo. Per questo motivo la leggenda sulla difficoltà del Linguaggio C ha preso piede.

Programmazione in C/C++

di Flavio Bernardotti – flavio@websitek.com - www.crackinguniversity2000.it

Livello: Principiante – Parte 1

Introduzione

Con l'andare avanti del tempo sono comparsi sul mercato un numero sempre maggiore di sistemi di sviluppo fino a giungere alle attuali distribuzioni in cui nei sistemi di sviluppo commercializzati troviamo tutti gli strumenti adatti ad affrontare tutte le problematiche che possono esserci nell'ambito dello sviluppo di programmi.
In altre parole, indipendentemente dal linguaggio, le problematiche che si riscontrano durente lo sviluppo di un programmo sono sempre le stesse e queste possono essere riassunte nelle seguenti :

Gestione interfaccia utente (form di I/O, dialog varie, ecc.)
Gestione I/O su files (gestione RAW files, gestione database)
Gestione output su periferiche diverse dal video (stampanti)

Dicevamo che fino a 10 anni fa, in ambiente DOS e Unix, l'unica soluzione era quella di acquistare diverse librerie ciascuna delle quali si interessava di risolvere un certo problema.
Ad esempio il sottoscritto utilizzava Vermont View per la gestione delle finestre video, Db Vista per la gestione degli archivi ed altre librerie per la gestione dei report.
L'avvento di Windows ha portato alla distribuzione con i normali sistemi di sviluppo delle API che permettevano le stesse gestioni, senza avere più la necessità di dover acquistare come moduli separati dal compilatore le varie librerie.
Gli elevati costi che anni fa possedevano questi moduli aveva portato molti a scriversi interamente le varie librerie.
Questo chiaramente era un fase complessa e per questo motivo nacque la favola delle complicazioni estreme del linguaggio C.
Al giorno d'oggi i vari linguaggi possiedono metodi comuni per la creazione di programmi, motivo per cui le differenze tra uno e l'altro di fatto non sono poi cosi grandi.
Oltre a questo le varie case hanno cercato nella creazione di tali basi comuni la chiave per la creazione di sistemi di sviluppo simili tanto da poter essere implementati nella stessa base funzionale.
Prendiamo ad esempio l'ambiente Visual Studio di Microsoft.
Nell'ambito della serie Visual troviamo :

```
Visual C++
Visual Basic
Visual Java
```

Dicevamo prima che le basi adatte a risolvere certi tipi di problemi sono uguali per cui anche I metodi di programmazione sono simili.
Prendiamo ad esempio le problematiche che dicevamo prima.
La gestione dei database viene risolta dal sistema ADO, DAO, OLE DB, ODBC i quali sono implementati in un certo modo nell'ambito di Windows per cui la metodologia di utilizzo dai vari Visual Basic, Visual C++ e Visual Java sono uguali addirittura a livello di metodi implementati dentro alle classi di gestione.
Il termine classi ci porta subito alla mente la programmazione Object Oriented la quale è sicuramente la chiave per tale standardizzazione, se cosi la possiamo chiamare.
La gestione dell'interfaccia utente viene svolta dalle API di Windows sulle quali tutti i linguaggi si basano.
All'interno di queste API troviamo anche altre funzioni indirizzate alla gestione di Windows e queste essendo appunto presenti in quest'ultimo possono essere utilizzate semplicemente tramite delle funzioni agganciate a queste.
Vedremo nel prossimo capitolo i concetti di base delle programmazione Object Oriented anche se comunque possiamo già da adesso utilizzare il concetto di oggetto per poter dire che nell'ambito di queste API di gestione di Windows troviamo tutte le funzioni per gestire i vari oggetti tra cui esistono anche quelli indirizzati alla gestione dell'interfaccia utente come ad esempio i pulsanti, i campi di edit, le liste ecc.

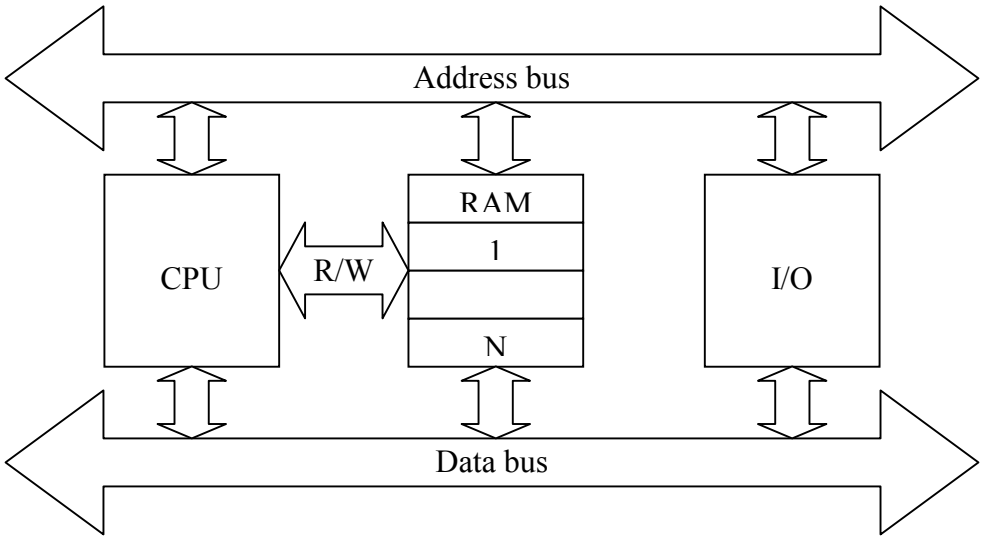
Per risolvere altri problemi come ad esempio i report su stampante con il sistema Visual Studio viene distribuito un pacchetto che permette la gestione visuale di questi e il loro successivo utilizzo tramite semplici funzioni di richiamo utilizzabili nei vari linguaggi.
Mi riferisco a Crystal Report della Segate software il quale, dopo la creazione a video del report e dopo l'attribuzione di un nome che gli permette di essere salvato su file, permette con una semplice funzione, PrintReport(), di richiamarlo.
In ogni caso anche se i concetti di programmazione si sono semplificati prima di poter mettersi li e scrivere un programma sono necessari alcuni concetti, alcuni dei quali generali legati alla struttura dei computers e dei sistemi operativi ed altri invece legati alla teoria del linguaggio e del sistema di sviluppo usato.
Questo corso informale di programmazione l'ho voluto creare per soddisfare la volontà di alcuni di dedicarsi allo sviluppo di software, volontà notata nell'ambito di alcune mailing list a cui aderisco in internet.
L'ho voluto chiamare corso informale in quanto in tutti gli anni in cui ho svolto degli incarichi di insegnamento ho notato che mettersi li a passare ore ed interi giorni a parlare di teoria pura serviva soltanto a confondere le idee.
Durante questi periodi ho notato che le persone necessitavano semplicemente di metodi che permettessero di creare una simbologia mentale di qualche cosa che di fatto è puramente astratto.
Approfondire molto i concetti in una fase iniziale serve solo a complicarsi la vita per cui il metodo migliore è quello di spiegare alcuni concetti di base passandoci sopra mediante spiegazioni metaforiche tendenti spesso a definire le celle di memoria come scatole di scarpe piuttosto che chip con all'interno sistemi di silicio adatti a mantenere segnali elettrici idonei a rappresentare gli stati logici di 0 e 1.

Il sistema che chiamiamo computer

Il sistema che definiamo normalmente con il termine di computer è semplicemente una macchina elettronica costituita da 5 blocchi logici funzionali, alcuni dei quali utilizzati per svolgere compiti di calcoli legati alla manipolazione di certe informazioni e altri invece adatti a fornire un metodo di dialogo con il mondo esterno al fine di ricevere e comunicare i dati necessari a queste elaborazioni.
I blocchi d fatto sono i seguenti :

CPU destinata a svolgere i calcoli aritmetici/logici nel'ambito del sistema
RAM/ROM ovvero la memoria che mantiene memorizzati i dati
I blocchi di I/O con sopra dei processori dedicati detinati al controllo delle periferiche di colloquio con il mondo esterno
Un ADDRESS Bus destinato alla comunicazione degli indirizzi interessati nelle operazioni svolte dalla CPU.
Un DATA Bus su cui corrono i dati da/per la CPU, memoria e blocchi di I/O

Volendo vedere il tutti disegnato avremmo :



Chiaramente questa è un schematizzazione superficiale ma sufficiente a fare capire come funziona un sistema.
In pratica la CPU contiene al suo interno un interprete in grado di capire e svolgere alcune operazioni, in genere calcoli matematici e logici.

Quando il sistema parte la CPU inserisce sul BUS INDIRIZZI l'indirizzo che vuole leggere, mette sul bus di EAD/WRITE (R/W) il segnale che indica alla memoria se l'indirizzo sull'address bus è il lettura o scrittura, la RAM prede questo valore e lo mette sul DATA BUS, la CU lo prende e lo interpreta.
La memoria la possiamo vedere come una serie N di contenitori numerati da 0 a N, dove N è la capacità in BYTES della memoria del nostro sistema, in ciascuno dei quali possiamo mantenere memorizzati dei valori numerici.
I computers utilizzano la metodologia binaria per la rappresentazione dei numeri grazie alla capacità di mantenere dei segnali elettrici attivi.
In pratica la combinazione di 8 segnali elettrici, ciascuno dei quali può essere a tensione a +5V oppure 0, permette di rappresentare numeri in formato binario.
Ogni segnale di questi 8 viene chiamato BIT mentre tutti e 8 sono quelli che chiamiamo BYTES.
Il sistema binari è solo un altro metodo per rappresentare con soli due simboli un valore numerico.
Lo spostamento verso sinistra di un BIT significa moltiplicarlo per 2.
Praticamente abbiamo

```
000 000 01  =     1
000 000 10  =     2
000 001 00  =     4
000 010 00  =     8
000 100 00  =     16
001 000 00  =     32
010 000 00  =     64
100 000 00  =     128
```

Per cui ogni posizione vale un valore.
Per fare la conversione è sufficiente sommare i valori posizionali di ogni singolo vaore a 1.
Ad esempio :

```
1
2 6 3 1
8 4 2 6 8 4 2 1
------------------
0 0 0 1 1 1 0 0    =  4 + 8 + 16 = 28
```

Ritornando ala memoria dicevamo che all'interno di questa sono contenuti numeri che possono essere semplici dati oppure possono essere codici di istruzioni che la CPU deve eseguire.
I costruttori delle CPU quando eseguono i progetti creano anche un codice che il processore è in grado di interpretare e eseguire.
Questi sono i vari codici operativi che i genere visti a livello di memoria sono dei semplici numeri in formato binario.
Ricordatevi che in memoria ci sono OPERAZIONI e DATI.
All'interno della CPU generalmente ci sono dei registri che possono essere paragonati a celle di memoria che il processore utilizza momentaneamente per delle operazioni.
Ad esempio un operazione eseguibile dal processore, visualizzata come numeri esadecmali, potrebbe essere la seguente :

```
C7 45 C4 40 00 00 00
```

Si tratta di 7 BYTES ciascun numero memorizzato dentro una cella di memoria.
Vedendo questo tipo di istruzioni capirete immediatamente come sarebbe complesso programmare se ci fosse solo questo metodo per istruire il processore.
Per semplificare la vita sono stati creati i linguaggi ad alto livello i quali grazie a dei programmi che dopo aver eseguito l'analisi sintattica e semantica di quanto da noi scritto, crea i codici operativi espressi appunto come sequenze di numeri corrispondenti ai codici operativi.
Tra i linguaggi a più alto livello troviamo anche l'assembler che normalmente viene considerato come il codice macchina anche se di fatto questo non è vero in quanto anche questo linguaggio è di fatto una rappresentazione mnemonica delle istruzioni binarie.
Chiaramente le istruzioni sono molto semplici e possono fare cose banali al contrario di linguaggi a più alto livello i quali in una sola istruzione possono svolgere anche compiti molto complessi.
Dicevamo prima che la CPU tramite i BUS colloquia con la memoria anche se di fatto lo stesso lavoro lo fa con le periferiche di I/O collegate tramite le apposite interfacce hardware.
Ad esempio un processore dedicato elabora i segnali dalla tastiera, un altro quelli della porta seriale e cosi via.

Praticamente il sistema legge delle operazioni in memoria, le svolge e quando riceve l'istruzione adatta le comunica ai blocchi di I/O per la gestione su un interfaccia destinata all'uscita verso il mondo esterno dei dati.
I codici di prima in esadecimale corrispondono all'istruzione assembler :

```
mov     [00407899], 40h
```

che dice :

MUOVI ALL'INDIRIZZO DI MEMORIA 00407899 IL VALORE 40 IN ESADECIMALE

In pratica dopo l'esecuzione di questa istruzione nella cella di memoria 00407899 si troverà il valore 40H.
Se volessimo fare in assembler un programma che calcola l'area di un quadrato e la visualizza video dovremmo fare :

```
COMUNICA AL GESTORE DELLA TASTIERA IL SERVIZIO LEGGI UN VALORE
LEGGI DA TASTIERA IL VALORE DEL LATO TRAMITE LA SCHEDA DI I/O
METTILO IN UN BYTE IN MEMORIA
PRENDI IL VALORE E METTOILO DENTRO ALL CPU IN UN REGISTRO
MOLTIPLICA IL VALORE DEL REGISTRO PER SE STESSO
PRENDI IL VALORE E METTILO IN MEMORIA
COMUNICA AL GESTORE VIDEO L'INTENZIONE DI STAMPARE UN VALORE
PASSA IL VALORE AL SISTEMA DI I/O CHE SCRIVE A VIDEO
```

Questa idea di usare la memoria ad indirizzi è quella che si deve avere sempre a mente anche quando si programma con un linguaggio come il C.
A questo livello avete capito che la memoria viene utilizzata per memorizzare dei valori.
In pratica ci sono istruzioni che dicono metti a questo indirizzo il valore ed altr fnzioni che dicono leggi da questa cella il valore.
Quando si devono memorizzare in memoria dei valori si deve sempre sapere dove questi sono messi e quanti bytes sono grandi.
In pratica valori compresi tra 0 e 255 possono essere memorizzati dentro a 1 BYTE.
Valori maggiori come ad esempio da 0 a 65536 possono invece stare dentro 2 BYTES.
Numeri fino a 4 miliardi invece stanno dentro a 4 BYTES.
Chiaramente è compito del programmatore dire dove una variabile deve essere memorizzata e quanto questa è grande.
Fortunatamente questo compito linguaggi a più alto livello lo fanno da soli.
Prima di andare avanti e iniziare a vedere i principi dei linguaggi è necessario capire ancora due cose.
Quando una persona a digiuno di programmazione vede un programma a video non riesce a capire come di fatto delle semplici operazioni matematiche possano di fatto ottenere quei risultati come ad esempio visualizzare immagini a video, comandare delle funzioni ecc.
Quando i costruttori di schede di I/O ne commercializzano na creano al loro interno delle porte tramite le quali è possibile programmare le funzionalità di queste.
Ad esempio la scheda che gestisce la porta seriale, quella del modem, possiede alcune porte tra cui la 0x2F4, la 0x2F8 ecc.
Inserendo dei valori dentro a queste, nel modo descritto dai costruttori, settiamo le specifiche come ad esempio la velocità di trasmissione, il numero di bits di dati e possiamo richiedere lo svolgimento di certe funzioni come ad esempio trasmetti un carattere.
Grazie a questi servizi i costruttori dei sistemi hanno inserito dentro a delle memorie non volatili (ROM, EPROM ecc.) quello che viene definito con il termine di BIOS ovvero Basic Input Output System.
Il BIOS è quello che offre alcuni servizi di base ad un livello un po' più alto rispetto alla programmazione diretta delle porte.
In pratica utilizzando delle sequenze di istruzioni su queste il BIOS offre delle funzioni destinate al controllo delle periferiche.
Le funzioni sono del tipo :

```
SCRIVI UN SETTORE DEL DISCO
SCRIVI N CARATTERE A VIDEO
LEGGI UN CARATTERE DALLA SERIALE
```

Per fare questo sono dettate delle specifiche come ad esempio :

per scrivere un carattere a video alla posizione X,Y mette il valore del carattere dentro al registro AL, il valore di X dentro al registro BL, quello di Y dentro al registro BH e poi chiamate l'interrupt 10H.
In pratica non specifichiamo a fondo che cosa sono gli interruptus (se volete approfondire potete leggere il mio libro 'Linguaggio C a basso livello del 1986) ma semplicemente diciamo che sno dei metodi per hamare delle funzioni inserite dentro al BIOS.
Capirete che le funzioni di BIOS sono proprio minimali e semplificano solo di poco quella che sarebbe la vita del programmatore rispetto la programmazione diretta delle porte hardware.
Basandosi su questi servizi BIOS quelli che hanno scritto i sistemi operativi hanno elevato ancora il livello delle funzioni rispetto le periferiche.
Mentre a livello di BIOS, ad esempio per quanto riguardava la programmazione del disco, trovavamo solo funzioni minimali come ad esempio scrivi un settore, a livello d sistema operativo troviamo funzioni come ad esempio scrivi questo buffer sul file XXXX oppure scrivi questa stringa a video usando gli attributi YYYY.
Quindi i sistemi operativi mettendosi di mezzo tra il sistema hardware e il mondo esterno offrono funzioni che semplificano la vita al programmatore.
A più alto livello ancora troviamo le funzioni di libreria dei linguaggi i quali basandosi spesso sulle funzioni del sistema operativo elevano ancora l'astrazione necessaria per rifarsi all'uso delle periferiche.
Capite che senza il controllo delle periferiche di I/O qualsiasi funzione fatta con il computer sarebbe inutile.
Dovendo dire solo due parole sulla gestione degli indirizzi fatti dal computer potremmo dire che se i sistemi fossero nati al giorno d'oggi l'indirizzamento di memoria sarebbe avvenuto utilizzando dei numeri a 32 BITS i quali, come abbiamo visto, possono rappresentare 4 miliardi di valori.
Purtroppo nei primi anni del 1980, e precisamente dal 1983, i sistemi possedevano limitazioni legate alla costituzione fisica dell'address bus il quale era di fatto 20 BITS e grazie ad un sistema particolare poteva indirizzare blocchi di memoria abbastanza grossi ma considerandoli mediante un indirizzo di partenza e un offset che poteva essere al massimo 65538 BYTES.
In altre parole se nel sistema avevate 640 KB il sistema poteva usare per l'indirizzamento un indirizzo qualsiasi in questo ambito e poi da questo indirizzo di partenza contare 64KB.
Questo ci ha portato per diversi anni a considerare gli indirizzi nella forma :

SEGMENTO:OFFSET

Infatti vi sarà sicuramente capitato di vedere indirizzi nella forma :

0040:4588

Come abbiamo detto, nei programmi a 16 BITS, ancora adesso utilizziamo questo metodo di indirizzamento il quale farà in modo che potremo avere variabili il cui indirizzo in memoria è di tipo NEAR ovvero rappresentato solo dal segmento ed altri in cui il tipo è FAR rappresentato da SEGMENTO:OFFSET.
Abbiamo d4etto prima che all'interno della CPU esistono dei registri i quali sono come delle celle di memoria all'interno del CHIP i quali possono essere usato come sistemi di memorizzazione temporanei, utilizzati nei calcoli dell CPU stessa, oppure possono possedere dei significati speciali come ad esempio quello di contenere dei valori di segmenti particolari come ad esempio il registro CS chiamato CODE SEGMENT, utilizzato per creare l'indirizzo reale nel caso di utilizzo di variabili NEAR, DS chiamato DATA SEGMENT ovvero segmento dati, SS per la STACK ovvero stack segment ecc.
Quando nei programmi a 16 BITS il sistema trova istruzioni assembler del tipo :

MOV [1234], 90h

Crea l'indirizzo reale con :

MOV [DS:1234], 90h

Lo stesso viene fatto per l'indentificazione della posizione dell'istruzione che deve essere eseguita.
Nel caso di un salto NEAR (vicino) la codifica avviene con :

JMP NEAR 1234

JMP CS:1234

Esiste , ereditato da questo metodo hardware, il riproporsi del concetto di segmentazione nell'ambito dei programmi.
In altre parole vedremo che concetti come lo STACK SEGMENT e altri viene riproposto nei concetti legati alla gestione di variabili e funzioni.

I linguaggi

A questo punto iniziamo a vedere che cosa servono i linguaggi di programmazione.
Un programma, a parte il fatto di eseguire delle serie di calcoli, gestisce un flusso di esecuzione in relazione al colloquio tenuto con l'utente.
In pratica, portando un gioco come esempio, ci saranno dei calcoli interni fatti dalla CPU destinati a creare le immagini visualizzate a video mediante funzioni di I/O sulla scheda grafica ma ci saranno anche altre funzioni destinate a leggere l'input da tastiera indirizzate a stabilire dove deve essere posizionato ad esempio l'immagine del personaggio del gioco stesso.
Quindi un linguaggio possiede istruzioni destinate ad elaborare dei valori contenuti dentro a delle variabili ma anche altre destinate a eseguire valutazioni indirizzate a gestire i flussi di elaborazione.
Ma che in cosa consiste programmare ?
Programmare significa creare i modelli matematici dei problemi che devono essere riprodotti su di un sistema informatico, mediante l'analisi di questi con la dichiarazione delle variabile dei dati descrittivi e mediante la creazione di tutte e istruzioni destinate a manipolare questi dati.
Da questo si comprende che la programmazione in genere può essere suddivisa in due parti ovvero :

- **La parte dichiarativa**
- **La parte algoritmica**

Nella prima fase si osservano i problemi, suddividendoli, se possibile, in problemi minori e quindi piu' semplici, identificando tutti i dati che sono necessari per la descrizione di questi.
Supponendo ce dobbiamo creare un programma che calcola le aree di parallelepipedi dovremo memorizzare da qualche parte i valori dei due lati necessari per l'esecuzione del calcolo.
Prima parlando di memoria avremmo potuto riservare alcuni BYTES per questo scopo.
In altre parole ci saremmo potuti scrivere da qualche parte che l'altezza del parallelepipedo veniva memorizzata all'indirizzo 000001000 e quella della larghezza all'indirizzo 00001002.
Nei linguaggi a più alto livello, nella fase dichiarativa, è possibile usare il concetto di variabile.
La dichiarazione di una variabile ci permetterà di riferirci successivamente ad un valore senza doverci preoccupare di dove il sistema ha memorizzato, come indirizzo d memoria, un certo valore.
In altre parole definendo la variabile altezza e quella larghezza potremo successivamente riferirci mediante il loro nome a queste ignorando di fatto a quale indirizzo il linguaggio ha memorizzato il tutto.
Prima avevamo anche detto che un valore possiede un indirizzo di dove questo viene memorizzato ma anche di una dimensione in bytes.
La dichiarazione di una variabile in linguaggio Cha la seguente sintassi :

classe di memoria tipo nome

Lasciamo perdere per ora la classe d memoria e vediamo solo le altre due specifiche.
Il nome è quello che ci permetterà di identificare un certo valore.
Normalmente può essere una stringa fino a 32 caratteri, anche se questo dipende dal compilatore.
Il tipo invece è quello che stabilisce la dimensione massima che può memorizzare la variabile.
I tipi sono :

```
char  (1 BYTE )    255 valori da -127 a +128
int   ( 2 BYTES ) 65536 valori da -32767 a +32768
long  ( 4 BYTES ) 4miliardi di valori da - 2 miliardi a +2 miliardi e rotti
```

Questi in linguaggio C sono da considerare proprio come sequenze di N BYTES.
Es.

```
Int   a = 12 = 000 011 00 byte 0001
                           byte 0002
```

0000
1100

Per questo motivo in questi casi non lasciatevi portare fuori dalla presenza di un tipo char in quanto questo è solo ideale per memorizare un carattere, in quanto questo è rappresentato da un valore ASCII compreso tra 0 e 255, ma ricordatevi sempre che in linguaggio C non esiste differenza tra un carattere ed un numero come magari in altri linguaggi come il basic esiste.
In C esistono altri tipi numerici che servono a mantenere i numeri memorizzati con il formato in virgola mobile.
Si tratta de tipi :

float
double

Mentre nei tipi di prima i valori potevano essere solo interi qui, grazie al formato di memorizzazione, possono essere anche con la virgola come ad esempio 6,899 oppure 2E3 in esponenziale.
Prima di vedere altri concetti legati alla dichiarazione delle variabili è necessario fare ancora un precisazione sui valori d prima.
Abbiamo detto che i valori sono compresi tra un valore negativo ed uno positivo.
In Linguaggio C esiste la specifica che è possibile mettere prima del tipo per dire al sistema che il valore è da considerare senza segno.
In pratica abbiamo detto che il valore poteva essere visto come il numero di bytes destinati al tipo.
Questo è solo parzialmente preciso in quanto per il valore vengono usati tutti i BITS meno uno che di fatto viene usato per il segno.
In altre parole ad esempio il tipo int possiede 15 bits per il valore e quindi può rappresentare 32768 numeri e un bit per il segno (+-).
Volendo usare anche questo bit per l valore è possibile usare prima del tipo la specifica unsigned.
Ad esempio un variabile :

unsigned int a

può memorizzare valori da 0 a 65536, tutti come valori positivi.
In questo caso i valori assegnabili diventano i seguenti.

```
unsigned char    -   da 0 a 255
unsigned int     -   da 0 a 65535
unsigned short   -   da 0 a 65535
unsigned long    -   da 0 a 4294967295
```

Se nei nostri programmi dobbiamo fare uso di numeri frazionari abbiamo a disposizione due tipi di variabili in virgola mobile e precisamente il float e il double.
L'occupazione in byte e il valore assegnabile a ciscun tipo sono precisamente :

```
float  -   4 bytes  da +- 1.701411E-38   a  +- 1.701411E38
double -   8 bytes  da +-      1.0E-307  a  +-      1.0E307
```

A questo punto vi ce abbiamo svito solo valori numeri ci vi chiederete come è possibile memorizzare delle stringhe intese come sequenze di caratteri ASCII.
Abbiamo detto prima che un carattere in C non è diverso da un numero tanto che possiamo anche fare delle cose del tipo :

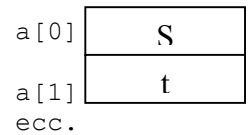
```
int a, b, c;            // Tre variabili intere

a = 25;                 // Ad a viene assegnato 25
b = 'B';                // b viene asegnato con 65 ovvero il codice numerico
ASCII di B

c = b - a;              // c vale 40 ovvero 'B' - 25 (65-25)
```

Mediante il concetto di array possiamo memorizzare delle squenze di valori per cui anche di caratteri.
La dichiarazione di array avviene mediante l'uso delle parentesi quadre.

```
int   a[20];
```

significa che a è una sequenza di 20 interi numerati da 0 a 19.
Per questo motivo possiamo fare :

```
char a[20] = "Stringa";
```



Molti linguaggi come il Basic trattano come oggetti le stringhe.

Il Linguaggio C non lo fa per cui anche funzioni semplici come la manipolazione di stringhe, copia, comprazione, concatenazione, vengono svolte da algoritmi.
Il basic permette ad esempio l'assegnazione tra dure variabili con :

```
a$ = "Stringa";
b$ = a$;
```

Il C dovrebbe eseguire un algoritmo del tipo :

inizializza un variabile d'indice a 0
segna qui l'inizio
prendi l'elemento indicato dalla variabile indice dentro all'array a
guarda se è l'ultimo carattere
se non lo è copialo dentro alla posizione indice di b , incrementa la variabile indice e vai all'inizio
se no finisci

Avete visto la domanda 'è l'ultimo carattere dell'array ?' ?
In C è necessario inserire alla fine delle sequenze di caratteri uno particolare che indica la fine della stringa.
Normalmente questo è il valore NULL ovvero lo 0.
In pratica il numero di caratteri necessari in un array destinato a memorizzare delle stringhe è sempre più grande di 1 carattere destinato al fine stringa.
Una stringa del tipo :

"ABCD"

non è lunga 4 caratteri ma bensì 5 in quanto in realtà la stringa è :

"ABCD0"

Fate attenzione in quanto l'inizializzazione diretta mette automaticamente il valore 0 alla fine.
Nel caso di dichiarazione di questo tipo, ovvero senza specificare la dimensione dell'array, il sistema autodimensiona l'array stesso alla lunghezza dels tringa + 1 carattere.

Char a[] = "ABCD";

Sarebbe come dire :

Char a[5] = "ABCD"; dove a[0] = 'A' , a[1] = 'B', a[2] = 'C', a[3] = 'D', a[4] = '\0'

Dimensionare con :

char a[20] = "ABCD";

significa riservare in memoria per la variabile a 20 bytes ma sul momento inseirgli solo 5 caratteri.
Dicevamo che il Linguaggio C non possiede istruzioni per la manipolazione di stringhe.
Queste sono implementate nelle librerie normalmente distribuite con il Linguaggio come normali funzioni.
Ad esempio dentro a queste librerie ci troviamo funzioni come strcpy (copia una stringa) strcmp (compara una stringa con un'altra), strcat (concatena due stringe).
L'assegnazione come abbiamo visto mette automaticamente il carattere di fine stringa.
Fate attenzione che magari in altri vasi sarà vostro compito mettere lo zero come terminatore.
Dimenticandosi di questo poterete bloccare l'esecuzione del programma in quanto l'algoritmo di copia aspettandosi di trovare il fine stringa continuerebbe all'infinito a copiare da una parte all'altra i valori.
Non abbiamo ancora parlato delle istruzioni ma essendo queste semplici vi voglio fare vedere come funziona una copia di una stringa.

```
Char a[] = "ABCD";
Char b[10];
Int indice = 0;
While(a[indice} != 0)
        B[indice] = a[indice];
```

Non trovando lo 0 il while non terminerebbe mai.
Ricordatevi che il C non esegue controllo di valori per cui se avessimo :

char a[2];
char b[2];

e poi copiassimo “ABCD” dentro ad a[] andremmo a finire con i caratteri in più dentro alla variabile dopo.
Infatti quello che capita è esattamente quello che capiterebbe in assembler con due variabili vicine in memoria.
L’utilizzo di ulteriori parentesi permetterebbe di definire delle matrici bidimensionali o anche a più dimensioni.
Ad esempio è possibile dichiarare variabili del tipo :

char m[10][5];

Detto in breve sarebbe come dire N righe di M colonne.
Prima di andare avanti ricordatevi sempre che le variabili qualsiasi esse siano sono sempre posizionate in memoria e quindi anche se di fatto l’indirizzo a cui si trovano noi lo possiamo quasi sempre ignorare questo c’e’ e dichiarare una variabile significa richiedere al compilatore di riservare per questa un certo numero di BYTES a partire da questo indirizzo di partenza.
Ricordatevi RISERVARE !!!
Altri linguaggi come il BASIC dopo aver riservato lo spazio controllano anche quello che si mette al suo interno avvisando se il contenuto non è idoneo per quella variabile.
Il C non lo fa e quindi a causa di errori di sottodimensionamento possono capitare errori che poi in fase di esecuzione diventano incomprensibili.
Facciamo un esempio che spesso capita al novizio.
Questo errore è quello legato al fatto di ignorare spesso il carattere di fine stringa e quindi di dimensionare la variabile solo per il numero di caratteri.
Dicevamo prima che lo STRING COPY (strcpy) è una funzione che si trova nella libreria standard del C che serve a copiare ogni singolo carattere di un array dentro ad un altro.
Come avevamo schematizzato prima l’algoritmo controllerebbe ogni carattere del primo array al fine di vedere se si è raggiunto il carattere di fine stringa (‘\0’ o NULL).
Ora facciamo questa ipotesi :

char m[4];
int x;

Ora se volessimo vedere in memoria come probabilmente verrebbero allocati gli oggetti avremmo :

Ora se facessimo :

strcpy(m, “ABCDE”)

il compilatore non ci darebbe errore in quanto la funzione direbbe :

COPIA “ABCDE” a partire dal primo indirizzo di dove si rova m.
Il carattere ‘E’ andrebbe afinire nel primo BYTE della variabile x mentre il carattere di fine stringa verrebbe messo nel secondo BYTE della variabile x.
Ora se noi dichiarassimo :

char k[10];

e poi usassimo la funzione per la copia di stringhe con :

strcpy(k,m);

E anche qui nessun problema in quanto l’agoritmo di strcpy prenderebbe il primo carattere di m, confronterebbe per vedere se è il fine stringa, non essendolo lo copierebbe nel primo byte di k,

incrementerebbe la variabile d'indice usata per accedere ad ogni singolo byte dell'array e continuerebbe fino a trovare il carattere di fine stringa e di fatto lo troverebbe a m[5].
Lo so che noi non lo abbiamo dichiarato con cosi tanti bytes ma il Linguaggio C usando un ottica a basso livello ragiona solo usando gli indirizzi per cui dire

m[5]

sarebbe come dire

l'indirizzo di m + il numero d'indice moltiplicato la dimensione del tipo.
Essendo il tipo char e quindi di dimensione 1 byte sarebbe come dire &m[0] + (5 * 1)
Avete visto il carattere & ?
In C significa 'l'indirizzo di' per cui in questo caso sarebbe come dire l'indirizzo di m[0].
Comunque tornando al caso di prima dicevamo che fino a questo punto non c'era nessun problema in quanto ci troveremmo con :

A	BYTE 1 di m ovvero m[0]
B	BYTE 2 di m ovvero m[1]
C	BYTE 3 di m ovvero m[2]
D	BYTE 4 di m ovvero m[3]
E	BYTE 1 di x
\0	BYTE 2 di x

Ora se aggiungessimo l'istruzione :

x = 25;

Cosa andremmo a fare ?
Semplice metteremmo dentro alla variabile x il valore 25 il quale però andrebbe a soprascrivere gli ultimi due caratteri dell'array.
Se dopo aver fatto questo usassimo nuovamente :

strcpy(k,m)

probabilmente ci troveremmo con il programma in crash in quanto l'algoritmo di strcpy non trovando più il carattere di fine stringa continuerebbe a copiare dentro alla zona di memoria di k correndo il rischio di arrivare a sconfinare in zone di memoria con contenuti critici.
Ricordatevi quindi di valutare sempre le dimensioni delle variabili in quanto il C non svolge nessun controllo ma eseguirebbe solo al funzione di definizione e basta.
Chiaramente qesta ottica ad indirizzi del Linguaggio C è la caratteristica che lo avvicina di più ai linguaggi a basso livello.
Per poter gestire questa filosofia degli indirizzi il Linguaggio C possiede un tipo aggiuntivo ovvero il tipo puntatore.
Che cosa è questo tipo ?
Bene.
Un tipo int, ad esempio, serve a dichiarare una variabile il cui contenuto è idoneo a contenere un valore numerico compreso tra –32767 e +32768.
Il tipo puntatore è il tipo idoneo a contenere un indirizzo di un oggetto.
La sua dichiarazione aviene con l'operatore unario * ovvero :

char *p;

Sarebbe come dire che p è idoneo a contenere l'indirizzo di un char.
Oppure :

int *p;

p è idoneo a contenere un indirizzo di un tipo intero.
Dopo aver dichiarato un tipo intero fate attenzione che il valore a cui punta non esiste fino a quando voi non gli assegnate un valore.
In pratica se fate :

char *c;

è come se diceste che c può contenere un indirizzo ma fno a questo punto non avete detto quale quindi lo spazio risulta non assegnato.
Successivamente facendo :

c = &m[0];

direste che lo spazio riservato all'indirizzo de punatore c viene assegnato a essere quello dell'indirizzo del primo byte dell'array m.
Potreste anche assegnare indirizzi direttamente con :

char *video = (char *) 0xB8000000L;

In pratica all'indirizzo 0xB8000000L c'era il buffer video dei monitor a caratteri DOS.
Questo avrebbe permesso di accedere direttamente a questa memoria.
Infatti i primi tempi che programmavo in GWBASIC usavo moltissimo le funzioni di PEEK e POKE le quali permettevano di leggere e scrivere direttamente in certe locazioni di memoria.
Passando al C sono stato per un certo periodo che dicevo : "ma possibile che con le 2000 fnzioni della libreria non ci siano le istruzioni di PEEK e POKE ?"
Certo.
A cosa servivano visto che con i puntatori potevo accedere direttamente alla memoria ?
Dopo aver fatto la dichiarazione :

char *p;

usare p senza il simbolo * significherebbe dire 'l'indirizzo contenuto in p' mentre usando :

*p

significherebbe dire 'il contenuto dell'indirizzo in p'.
Uno string copy usando gli indici poteva essere scritto con :

```
char a[] = "ABCD";
char b[5];
int i = 0;
while(a[I] != '\0') {
      b[I] = a[I];
      ++i;
}
b[I] = a[I];
```

Gli array vengono già trattati come indirizzi.
Usare

Char a[] = "ABCD";

Potrebbe essere dichiarato con I puntatori come :

char *a = "ABCD";

Volendo usare con lo strcpy I puntatori avremmo :

```
char *a = "ABCD";
char *n;
char b[5] ;
n = &b[0];
while(*a != '\0') {     // Fino a quando l'oggetto puntato da *a
      *n = *a;          // L'oggetto puntato da n e' uguale a quello puntato d a
            ++n;
      ++a;              // Incrementa sia n che a
}
*n = *a;
```

Ora però voi vi chiederete che differenza c'e' tra l'indirizzo di un intero da quello di un char.
Nessuno, ma la dichiarazione di un certo tipo fa si che l'aritmetica dei puntatori dimensioni in modo corretto i valori.
Cosa significa ?
Prendete l'esempio :

char *p;

Ora facendo :

++p

vorrebbe dire 'incrementa di 1 BYTE' l'indirizzo p .
Questo perche' l calcolo nell'aritmetica è :

indirizzo + N * dimensione_del_tipo

Qundi volendo vedere in bytes rispetto all'indirizzo di partenza quanto viene spostato avremmo :

```
int *a;
…
a = a + 1;
```

sarebbe

a = a + 1 * 2BYTES

2BYTES perché la dimensione del int è 2 BYTES.
Se avessimo :

```
long *p;
…
p = p + 2;
```

p = p + (2 * 4BYTES)

L'errore più comune di chi inizia è quello di usare I puntatori senza prima assegnargli a quali indirizzi puntano.
Se fate :

long *p

ricordatevi adi assegnargli qualche indirizzo statico di qualche cosa definito come ad esempio :

```
long *p;
…
long m[10];
p = &m[4];
```

In linguaggi C esistono I CAST ovvero le forzature che permettono di assegnare oggetti di tipi differenti a certi altri.
La forzatura permette di fare in modo che l'oggetto utilizzi poi le proprietà dell'oggetto assegnato.
Ad esempio è possibile fare :

```
char array[10];
int *p;
```

p = (char *) p;

In ogni caso no demoralizzatevi se non capite bene l'uso dei puntatori in quanto all'inizio potrete anche ignorarli.

Sicuramente il loro utilizzo permette di raggiungere prestazioni che solo linguaggi come il C o l'assembler possono avere, ma come abbiamo detto il loro uso potrà essere eseguito in un secondo tempo, quando sarete più ferrati sul linguaggio.
Per quanto riguarda gli indirizzi per ora ricordatevi solo che gli array senza specifica del singolo elemento sono considerati indirizzi.
Coma abbiamo già detto dopo avere fatto la dichiarazione :

char a[20];

scrivere

a oppure &a[0] è la stessa cosa.
In altre parole specificare

a

sarebbe come dire l'indirizzo di dove si trova l'array a, e quindi essendo a[0] il primo elemento dell'array lo stesso indirizzo lo si può avere dicendo &a[0].
In questa parte abbiamo accennato alla dichiarazione delle variabili ovvero di quella parte di lavoro che avviene nella parte dichiarativa della programmazione.
In questa si osservano i problemi, eali od astratti che siano, e si ricavano tutti i dati che possono esserci utili per raggiungere uno scopo.
Se dovessimo, ad esempio, creare un programma per memorizzare i dati di qualche cliente potremmo fare :

```
char  nome[40];
char cognome[40];
char indirizzo[40];
```

Ora all'interno di un programma potremmo avere moli di dati anche notevolmente grandi per cui potremmo trovarci ad avere la necessità di incapsularli al fine di definire a chi appartengono questi.
Il linguaggio C permette la definizione di quella che è la struttura.
Questa altro non è che un contenitore virtuale, identificato da un nome, il quale può essere successivamente utilizzato per dichiarare delle variabili le quali conterranno tutti i campi inseriti in fase di definizione.
Nell'esempio di prima i dati erano relativi ad un cliente per cui, utilizzando il concetto di struttura, avremmo potuto fare :

```
struct clienti {
        char  nome[40];
        char cognome[40];
        char indirizzo[40];
};
```

Fate attenzione che fare quello che abbiamo appena fatto non significa occupare uno spazio in memoria.
Questo verrà invece allocato quando useremo questa struttura per dichiarare una variabile.

struct clienti varCli;

varCli sarà allocata in memoria e occuperà esattamente la somma dei campi dichiarati dentro alla struttura stessa.
L'accesso ai campi avviene utilizzando il PUNTO (.) nel seguente modo :

```
varCli.nome
varCli.cognome
varCli.indirizzo
```

Capite che si potrebbe in questo modo dichiarare più varabili o addirittura array di strutture.
Anche per le strutture vale il discorso dei punatori.
Ad esempio potremmo dichiarare :

struct clienti *varCli;

Quanto spazio occuperebbe una dichiarazione di questo tipo ?
Esattamente quello che occuperebbe un indirizzo ovvero in genere o 2 BYTES o 4 BYTES.

Abbiamo detto appena poco prima che l'accesso ai membri avviene tramite i punti (.).
Nel caso in cui l'accesso sia a membri tramite puntatori a strutture, il tutto avviene inserendo i simboli -> tra il nome del puntatore e quello dei membri.

varCli->nome;
varCli->cognome;

Ricordatevi sempre che dichiarare un puntatore non significa allocare lo spazio per il tipo a cui punta ma solo lo spazio che conterrà quell'indirizzo.
Fino ad ora vi ho solo mostrato come è possibile dichiarare un puntatore e poi assegnarlo all'indirizzo di una variabile dichiarata da qualche altra parte.
In ogni caso questo non è l'unico mezzo per assegnare un spazio di memoria riservato ad un puntatore in quanto esistono dei casi in cui non essendo possibile a priori di sapere quanta memoria si deve allocare, si esegue l'allocazione dinamica tramite funzioni presenti dentro alle librerie del linguaggio C.
Non discuteremo di queste in quanto non sarebbe sufficiente lo spazio di questo volumetto, ma solo per farvi un esempio se noi sappiamo fin dall'inizio che un buffer potrà essere , ad esempio, 5000 bytes, potremo fare:

char buffer[5000];
….
char *p;
p = buffer;

oppure se preferite :

p = &buffer[0];

che è la stessa cosa.
Non sapendo la dimensione inizialmente potremo fare, al momento in cui veniamo a conoscenza di questa :

char *p;
….
p = malloc(10000);

In pratica la funzione malloc allocherebbe in memoria il numero di bytes specificato, restituirebbe l'indirizzo di dove si trova il primo byte il quale verrebbe assegnato al puntatore.
Una cosa su cui vale la pena soffermarsi sono i cast di cui avevamo detto prima qualche cosa.
In pratica un certo tipo possiede tutte le proprietà di questo ed in particolar modo quando si parla di indirizzi.
Prima di vedere ancora l'uso dei cast sugli indirizzi vediamoli utilizzati su tipologie normali.
Facciamo un esempio :

long n;
int p = 2;

Volendo assegnare il valore in p ad n potremmo forzare l'intero ad essere vistto come un long facendo :

m = (long) p;

In questo caso non capiterebbe nulla in quanto il long di destinazione è più grande dell'intero, come numero di bytes, per cui l'intero contenuto verrebbe inserito dentro alla variabile di destinazione.
Può capitare anche il caso contrario ovvero quello in cui un valore contenuto dentro ad una variabile maggiore viene inserito dentro ad una più piccola.

long m = 300000L;
int a;

a = (int) m;

In questo caso il contenuto verrebbe troncato.
Per capire come considerate il numero come se fosse in binario all'interno dei 4 BYTES del long e solo i primi 2 BYTES verrebbero assegnati alla variabile intera.
In questi casi i cast servirebbero solo ad adattare i valori ai contenuti di altri tipi di variabili.
Nal caso di indirizzi invece il cast è potente e permette di fare cose veramente notevoli.

Prendiamo l'esempio di una struttura :

```
struct clienti {
        char  nome[40];
        char cognome[40];
        char indirizzo[40];
};

struct clienti p;
```

Questo allocherebbe per la variabile p la dimensione in bytes data dalla somma dei campi ovvero 120 BYTES.
Ora se noi volessimo accedere BYTE a BYTE potremmo fare una cosa come segue :

```
char *m;

m = (char *) &p;
```

m è un puntatore e quindi può contenere un indirizzo di un tipo char.
&p restituisce l'indirizzo della struttura clienti p.
il cast (char *) forzerebbe questo indirizzo a essere consiederato come se di fatto fosse l'indirizzo di un array di char.
In questo modo facendo :

```
*m = 'A';
```

andremmo a mettere nel primo BYTE della struttura il carattere 'A'.
Incrementando l'indirizzo di 1 faremmo puntare m al byte successivo.
Una cosa che avevo tralasciato ma su cui possiamo aprire una nota veloce è quella relativa agli incrementi.
Fino ad adesso per incrementare di 1 una variabile facevamo :

```
a = a + 1;
```

Un metodo alternativo per l'incremento unitario è quello di fare :

a++ oppure ++a

La differenza tra il primo è il secondo si osserva solo nall'ambito delle priorità di operazioni al'interno di un'altra operazione del tipo :

```
b = a++
```

oppure

```
b = ++a
```

In un caso prima avviene l'assegnazione e poi l'incremento mentre nel secondo prima avviene l'incremento e poi l'assegnazione.
In altri casi dove l'incremento deve essere maggiore di uno o dove l'operazione è diversa dall'incremento è possibile fare :

```
b += 24; oppure b- =10; o ancora b *=3;
```

Concludiamo ora l'argomento dei puntatori ricordandoci di una cosa di cui dovrete sempre ricordarvi quando utilizzerete dei punatori.

QUANDO DICHIARATE UN PUNTATORE CHIEDETEVI SEMPRE PRIMA DI UTILIZZARLO SE CONTIENE UN INDIRIZZO O SE DOVETE ANCORA ASSEGNARGLIELO MEDIANTE L'ASSEGNAZIONE DI UN INDIRIZZO RELATIVO AD UNA VARIABILE ALLOCATA NORMALMENTE O MEDIANTE ALLOCAZIONE DINAMICA.

Un particolare metodo relativo all'utilizzo delle strutture è legato a quelle definite con il termine di strutture bit fields ovvero quelle in cui i vari campi sono costituiti da uno o più bits.

Prendiamo il caso di flags o indicatori di eventi particolari in cui utilizzare una variabile numerica sarebbe sciupata, anche se di fatto il loro uso non è legato all'economia di memoria.
Il concetto di struttura permetterebbe di incapsulare tutti i flags al suo interno.
Un esempio potrebbe essere :

```
struct flags {
        unsigned char bit1 : 1;
        unsigned char bit2 : 1;
        unsigned char bit3 : 1;
        unsigned char bit4 : 1;
        unsigned char bit5 : 1;
        unsigned char bit6 : 1;
        unsigned char bit7 : 1;
        unsigned char bit8 : 1;
};
```

Le variabili interne sono definite come unsigned char in quanto anche se di fatto possiamo trattarle a bits di fatto l'allocazione delle struttura terrà al minimo la dimensione di un unsigned char o comunque di un multiplo di questo.
Chiaramente i valori dei vari mebri potranno essere rappresentati dai valori che è possibile usare con il numero di bits specificato.
I numero di bits può essere anche differente da 1 bit.
Ad esempio potrebbe essere anche valida una dichiarazione del tipo :

```
struct flags {
        unsigned char bit1 : 4;
        unsigned char bit2 : 4;
};
```

L'uso di simili strutture potrebbe essere quello di memorizzare in ogni membro un flag con il fine di indicare un determinato stato all'interno del programma ma nessuno ci viete di utilizzarlo per altri scopi.
Ad esempio se noi dichiarassimo :

```
struct flags strBits;
```

Questo allocherebbe in memoria da qualche parte lo spazio relativo ad un unsigned char ovvero 1 BYTE.
Se successivamente dichiarassimo un punatore :

```
char *carattere;
```

avremmo dichiarato lo spazio sufficiente mantenere l'indirizzo che punta ad un tipo char.
Ora potremmo forzare con un cast l'indirizzo della struttura ad essere accettato dal punatore.

```
carattere = (char *) &strBits;
```

Ora se assegnassimo al valore puntato dal punatore un valore, ad esempio 'A' , con :

```
*carattere = 'A';
```

andremmo ad inserire all'interno dell'indirizzo del puntatore carattere il valore 'A'.
Ma questo è l'indirizzo di dove è stata allocata la struttura per cui di fatto se riprendessimo ad usare la struttura per se stessa potremmo andare a vedere i singoli bits del byte occupato con :

```
strBits.bit1
strBits.bit2
...
strBits.bitn
```

Il codice ASCII di 'A' è 65 per cui dentro al BYTE puntato da carattere avremo :

0	1	0	0	0	0	0	1

Infatti facendo un piccolo programmino che stampa il contenuto di ogni singolo campo della struttura avremmo la stampa a video del valore in formato binario :

```
#include <stdio.h>

struct flags {
        unsigned char bit1 : 1;
        unsigned char bit2 : 1;
        unsigned char bit3 : 1;
        unsigned char bit4 : 1;
        unsigned char bit5 : 1;
        unsigned char bit6 : 1;
        unsigned char bit7 : 1;
        unsigned char bit8 : 1;
};

struct flags strBits;

char *carattere;

void main(void)
{
        carattere = (char *) &strBits;

        *carattere = 'A';

        printf("\n%d%d%d%d%d%d%d%d",
        strBits.bit8, strBits.bit7, strBits.bit6, strBits.bit5,
        strBits.bit4, strBits.bit3, strBits.bit2, strBits.bit1);

}
```

Ora provate solo ad assegnare 1 un bit della struttura che prima era 0, ad esempio il bit 2 e poi riprovate stampare solo il valore puntato da carattere :

```
strBits.bit2 = 1;
```

Avremmo quindi :

0	1	0	0	0	0	1	1

Che in decimale equivale a 67 ovvero al carattere ‘C’.

Inserite in coda al programma, dopo l’assegnazione di prima :

```
printf(“%c”, *carattere);
```

Avrete C stampato a video.
Prima di poter parlare dell’altra specifica della dichiarazione, la classe d memoria, dobbiamo vedere la seconda parte di una programma ovvero la parte algoritmica.
Abbiamo detto che un programma si suddivide in due parti ovvero la parte dichiarativa e quella algoritmica.
Nella seconda parte grazie a istruzioni , intese come parole chiave del linguaggio, è possibile creare i flussi di esecuzione in cui le istruzioni eseguite sono a seguito di valutazioni fatte mano mano.
Le parole chiave del C sono :

```
int             char            float
double          struct          union
long            short           unsigned
auto            extern          register
typedef         static          goto
return          sizeof          break
continue        if              else
```

```
for             do              while
switch          case            default
```

La parte algoritmica

Fino ad adesso abbiamo osservato i problemi e abbiamo definito i dati che utilizzeremo dentro al programma.
A questo punto dobbiamo iniziare a vedere quella parte in cui si scrive il codice necessario a manipolare i dati per ottenere quello che ci prefiggiamo di fare.
Questa parte viene svolta dalla parte algoritmica dove usando semplici assegnazioni o funzioni delle librerie del linguaggio C manipoleremo i dati e li visualizzeremo a video, li scriveremo su files, creeremo dei rapporti di stampa o comunque faremo tutto quello che è necessario.
Il linguaggio C possiede poche parole chiave che sono quelle che abbiamo appena visto.
In ogni caso la fase di creazione di un programma eseguibile si suddivide in due parti ovvero nella compilazione, dove avviene un analisi sintattica e semantica e quindi una traduzione in un codice oggetto e successivamente nel link ovvero quella fase in cui questo codice oggetto viene analizzato e tutti i riferimenti simbolici vengono tradotti in riferimenti assoluti.
Cosa significa questo ?
Beh.. semplicemente che come abbiamo visto noi per riferirci, ad esempio, ad una variabile usiamo un nome ben definito che ci ricorda dello scopo di questa.
In altre parole noi usiamo nomi di variabili come nominativo, indirizzo ecc. ma poi in effetti a livelo di codice assemblativo i riferimenti verranno fatti solo per indirizzi.
Il link non solo esegue questa conversione ma unisce al nostro programma quel codice relativo alle funzioni di libreria che noi abbiamo utilizzato e che come codice sono presenti i files distribuiti con il compilatore.
Dicevamo che il Linguaggio C non tratta le stringhe ma che comunque, essendo un problema comune quello di manipolarle, all'interno di una libreria sono presenti tutte quelle funzioni adatte a svolgere questi compiti.
Non sarà quindi necessari scriverle ogni volta e vi assicuro che di funzioni ce ne sono fin quante ne volete.
Il segreto per imparare ad usarle è quello di mettersi a mente il compito che si vuole fare e di cercare nei gruppi funzionali delle descrizioni quelle funzioni che si adattano meglio a quel compito.
Esistono funzioni per la gestione della memoria, per la gestione dell' I/O, per la manipolazione di stringhe ecc. ecc.
Quindi da questo darebbe già possibile comprendere che il C è un linguaggio funzionale o procedurale.
Che cosa significa ?
Semplicemente che le varie istruzioni possono essere incapsulate all'interno di moduli chiamate funzioni le quali possiedono un nome che gli permette di essere richiamate in qualsiasi punto del programma.
I primi linguaggi di programmazione identificavano con un numero di linea ogni istruzione e poi in base a dei controlli fatti sui valori delle variabili si eseguivano dei salti a determinati punti.
Questi tipi di salti potevano essere condizionati od incondizionati, ovvero il linguaggio nel primo caso prima di eseguire il salto si memorizzava il punto di partenza in modo tale che a seguito di una certa istruzione di ritorno questo veniva ripristinato.
Chiaramente questo metodo di programmazione era confusionario in quanto era complesso tenere a mente a quali scopi erano state scritte determinati gruppi di istruzioni.
La creazione del concetto di funzione ha permesso di includere dentro a dei limiti definiti le funzioni indirizzate allo svolgimento di un certo compito.
La dichiarazione di una funzione avviene con :

classe_di memoria tipo_restituito nome_funzione(argomenti1, argomento 2, ecc.)

L'inizio e la fine del blocco contenente le istruzioni relative ad una determinata funzione è contrassegnato dalla parentesi graffa aperta e chiusa.
Ad esempio :

```
int     funzione(int argomento)
{
        int a;
        a = a + 1;
        return a;
}
```

La classe di memoria l'avevamo tralasciata prima con le variabili e la saltiamo anche adesso riproponendoci di parlarne dopo.
Una funzione può ricevere un certo numero di variabili come argomenti e restituire un valore di ritorno.

Ad esempio è una dichiarazione di funzione valida :

void funzione(int valore)

Questo dice che la funzione non restituisce nulla (void) e che riceve in ingresso una variabile intera.
Quando si parla di analisi in relazione al fatto di identificare i problemi e di suddividerli in sottoproblemi al fine di rendere più semplice la sua soluzione si può vedere la stessa cosa nell'ambito di un programma creato e strutturato mediante l'utilizzo di funzioni.
Abbiamo già detto che una funzione altro non è che un blocco di istruzioni che vengono eseguite richiamando il nome con cui abbiamo chiamato la funzione stessa.
Avendo la capacità di creare delle funzioni in modo tale da poter essere riutilizzate in altri ambiti, il linguaggio C dà la possibilità di inserirle, una volta compilate, dentro a delle librerie e quindi poi di poterle riutilizzare senza doverle riscrivere tutte le volte.
Il compilatore C viene fornito di un numero elevatissimo di funzioni di libreria indirizzate a svolgere i compiti più diversi.
Richiedete l'help del compilatore e vedrete che, al fine di dargli un ordine, le funzoni sono state raggruppate per tipo.
Ad esempio quelle che gestiscono le stringhe, quelle relative a funzioni matematiche, quelle utilizzate per l'input/output ecc.
Per potervi dimenticare in mezzo al grosso numero tenete sempre ben presente quello che volete fare e cercate nel gruppo idoneo.
Ad esempio se possedete un numero float, 5.89 ad esempio, e lo volete trasformare in un valore stringa, "5.89", andate a vedere nell'ambito delle funzioni di conversione e troverete la funzione che fa per voi, ftoa() in questo caso.
Prima di proseguire è necessario avere ben presente il processo eseguito da compilatore e dal link.
Quando scriviamo un programma in Linguaggio C utilizziamo dei riferimenti simbolici per manipolare le variabili e per richiamare le funzioni.
In altre parole le variabili, pur essendo memorizzate a certi indirizzi di memoria, possiedono un nome che noi utilizzeremo per la manipolazione del loro valore.
Ad esempio faremo :

a = 25;

Sapendo che in effetti la variabile a è memorizzata all'indirizzo 00405678, ad esempio, potremmo identificare in assembler l'istruzione di assegnazione con :

mov [00405678], 25

Lo stesso dicasi per il richiamo di funzioni.
Noi utilizziamo per rischiarle il nome che gli abbiamo assegnato in fase di scrittura del programma.

funzione(23)

Anche questa funzione possiede al prima istruzione del codice a lei interna ad un certo indirizzo.
In asembler, supponendo che l'indirizzo della funzione sia 00405678, avremmo :

call 00405678

Il compilatore esegue un analisi sintattica e semantica per vedere se il programma è stato scritto correttamente e poi crea per ogni file .C un file .OBJ.
Questo file .OBJ contiene ancora riferimenti simbolici.
Il linker eseguirà l'ultima fase della creazione del programma ovvero quella in cui i vari files .OBJ vengono presi e i loro riferimenti interni tradotti i riferimenti assoluti.
Dopo averli uniti viene creato il file eseguibile :EXE.
Al fine di strutturare in modo più ordinato un programma è possibile scrivere il codice in più files .C, ovvero file sorgente.
Dopo averli creati è possibile compilarli ed linkarli insieme alla fine.
I moduli .OBJ possono anche essere inseriti all'interno di librerie le quali possono essere unite in fase di link al programma.
Pensando all'esecuzione di un programma viene da chiedersi una cosa.
Da dove inizia l'esecuzione di un programma ?
I programmi scritti in C per convenzione iniziano l'esecuzione da quella che viene definita con il termine di main function.

In pratica in un programma esisterà una funzione che si chiama :

main()

Nel caso di Windows la funzione principale si chiama WinMain().
In altre parole il punto costituito dalla funzione main() o WinMain() verrà utilizzata come entry point.
Riprendiamo l'esempio di cui abbiamo parlato relativo alle funzioni per la manipolazione delle stringhe ovvero :

strcpy(), strcat(), strcmp()

Queste sono inserite dentro ad una libreria distribuita con tutti I compilatori C.
Nei nostri programmi utilizzeremo le funzioni ma non riscriveremo il codice in quanto queste verranno estratte dalle librerie ed inserite dentro al nostro programma dal link.
Nel linguaggio C esiste un problema dato dal compilatore e dal come questo analizza il sorgente.
In pratica l'analisi avviene dall'inizio del file per cui si verifica un problema.
In altre parole il compilatore potrebbe incontrare il richiamo di una funzione presente in una libreria o scritta solo successivamente all'interno del programma.
Il compilatore quando incontra un richiamo dovrebbe sapere quali sono le caratteristiche delle funzione ovvero quali parametri accetta e quali valori ritorna.
Per fere questo, se non è possibile scrivere la funzione prima di richiamarla, è sufficiente creare un prototipo che altro non è che la testata delle funzione in cui è possibile vedere tutte le caratteristiche della funzione stessa.
Ad esempio si potrebbe scrivere :

```
int     somma(int v1, int v2);

int     main(void)
{
        int c ;
        c = somma(22,89) ;
        printf(« %d », c) ;
}

int     somma(int v1, int v2)
{
        return v1 + v2 ;
}
```

I questo modo quando il compilatore incontrerà il richiamo alla funzione somma saprà già le caratteristiche di questa.
Se la funzione fosse scritta prima del suo richiamo, il prototipo non sarebbe necessario in quanto questa direbbe al compilatore quali sono le sue caratteristiche.
Se non mettete il prototipo il compilatore vi avvertirà che considererà la funzione come se ritornasse un intero.
Questo andrebbe bene nel caso in cui la funzione non restituisse nulla o se il valore reso fosse davvero un intero, ma creerebbe dei problemi in caso contrario.
I prototipi delle funzioni sono necessari anche per le funzioni di libreria ed è di fatto per questo motivo che insieme alle librerie vengono forniti dei files di include con estensione .H
Dentro ai files . H sono dichiarate delle costanti usate dai programmi, i prototipi delle funzioni relative alle loro librerie e certe variabili.
Anche i questo caso i files d'include raggruppano i dati per tipo.
Ad esempio il file stdio.h contiene le dichiarazioni relative allo standard d'input/output.
Questi devono essere importati nel sorgente grazie ad una specifica al preprocessore chiamata :

#include

Ad esempio le funzioni di elaborazione delle stringhe sono contenute dentro al file string.h il quale deve essere incluso con :

#include <stdio.h>

Il compilatore possiede delle directory di default dove sono inseriti i files di libreria ed un'altra directory dove sono presenti i files d'include.
Il fatto di specificare il nome del file tra < e > sarebbe come dire di cercarlo dentro alla directory di default degli include files.
Specificando invece con :

#include "stdio.h"

dice di cercare il file dentro alla directory corrente del progetto.
Ho definito come istruzione per il precompilatore in quanto tutte le istruzioni precedute da # sono specifiche che indirizzano il metodo di compilazione.
Ce ne sono altre come ad esempio per creare blocchi condizionali che vengono inseriti e compilati solo a seguito di certe valutazioni.
Una di queste, fondamentale, è quella che permette di definire delle costanti ovvero il costrutto :

#define

Ad esempio è possibile fare :

#define PGRECO 3.142

Attenzione che le costanti vengono prese nella fase di precompilazione, ovvero una fase prima della compilazione vera e propria, e vengono sostituite con i valori associati.
In pratica quando il compilatore troverà la specifica PGRECO la sostituirà brutalmente con 3.142.
E' anche possibile specificare forme più complesse come ad esempio :

#define SETTOZERO(x,len) (memset((char *)x, 0, len))

Ricordatevi che le #define vengono sostituite.
In ogni caso tralasceremo per questioni di spazio nell'ambito di questo volumetto.
Ora che abbiamo parlato di funzioni è stato definito un certo numero di zone dove possono avvenire certe cose.
Che cosa significa ?
Semplicemente che prima non potevamo parlare di punti in cui avvengono le dichiarazioni ora invece possiamo riferirci a variabili definite fuori dalle funzioni e dentro.
Una variabile possiede due caratteristiche particolari che sono la visibilità e il ciclo di vita.
La prima caratteristica stabilisce dove questa variabile può essere vista.
Se la variabile è dichiarata fuori delle funzioni la sua visibilità è globale ovvero tutte le funzioni del programma la possono vedere e quindi questa può essere utilizzata in qualsiasi punto.
Per ora non parliamo ancora di moduli creati da files .c differenti.
Il ciclo di vita è invece quella caratteristica che ci dice per quanto tempo la variabile esisterà.
Se dichiarata al di fuori delle funzioni la sua vita sarà per tutta la durata del programma mentre se questa viene dichiarata dentro ad una funzione questa esisterà solo per il tempo di durata della funzione.
Volendo dare un occhiata più a fondo dovremo vedere l'allocazione dal punto di vista dei segmenti.
In pratica le dichiarazioni di variabili globali avvengono dentro al DATA SEGMENT mentre quelle dentro alle funzioni sono allocate dentro allo STACK SEGMENT.
Fate attenzione che il compilatore dimensiona lo stack ad una certa dimensione di default in genere 4096 BYTES per cui sapendo che le dichiarazioni delle variabili dentro alle funzioni occupa lo spazio dentro allo stack segment, il loro dimensionamento troppo grosso potrebbe creare problemi di STACK OVERFLOW (avete mai visto questo messaggio ?).
In altre parole se lo stack è di 4096 BYTES una cosa del tipo :

```
int funzione()
{
        char a[9000];
```

creerebbe uno stack overflow.
La specifica statica relativa alla classe di memoria farebbe si che la variabile non verrebbe più alocata nello stack ma in una zona di memoria appunto statica.
La dichiarazione si trasformerebbe in :

```
int funzione()
{
```

static char a[9000];

Fate attenzione che le varibili locali dentro allo stack perdono il loro contenuto ogni volta che il programma esce dalla funzione per cui anche se da questa passerete più volte durante l'esecuzione, dovrete assegnare il valore ogni volta.
Se la variabile è definita come statica invece manterrà il valore.
Sapete che cosa è lo stack ?
In pratica è come la pila delle consumazioni del barista.
L'ultimo biglietto inserito sarà il primo ad uscire in fase di estrazione.
Il sistema utilizza lo stack per diversi motivi.
Uno, coma abbiamo visto, è quello di allocarci dentro le variabili locali.
Il secondo è quello di inserirgli dentro i valori di ritorno al momento della chiamata ad una funzione.
Alla chiamata di una funzione il sistema prende l'indirizzo successivo a quello della chiamata stessa e lo inserisce dentro allo stack.
Quando incontrerà un istruzione di return (RET) preleverà questo indirizzo estraendolo dallo stack e lo ripristinerà.
Quando vengono utilizzate variabili locali e quando quelle globali ?
Questo è vostro compito valutarlo.
In pratica se una variabile ha solo scopo nell'ambito di una funzione allora dichiaratela all'interno di questa.
Se la variabile deve essere vista da qualsiasi funzione o almeno da piu' di una funzione allora dichiaratela globalmente.
Una variabile possiede oltre all'identificatore ed a un tipo anche una classe di memorizzazione.
Le variabili, se non dichiarate altrimenti, vengono considerate come automatiche e valgono per queste le regole appena viste.
Un ulteriore classe di memorizzazione e' costituita da quelle definite come statiche per le quali viene allocata una memoria privata nella funzione in cui viene dichiarata.
Mentre una variabile auto perde il valore all'uscita dalla funzione, la variabile statica lo mantiene Inalterato.
Un esempio di dichiarazione

static char alpha;

Nel caso in cui si faccia un ampio uso di una variabile automatica esiste la classe di memorizzazione definita register.
Questa classe non accetta tutti i tipi ma solo i char, gli int e i puntatori che vedremo a suo tempo.
Una dichiarazione di questo tipo fa si che l'allocazione avvenga in un registro, ottenendo in questo modo un codice estremamente veloce e compatto.
Per le variabili register esiste la possibilita' di richiedere l'allocazione per sole due variabili per ogni funzione.
Richiedere non significa in ogni caso ottenere.
Nel caso che non sia possibile ottenere questa classe di memorizzazione la variabile viene trasformata in una semplice variabile automatica.
Fate attenzione che comunque la classe register appartiene alle variabili automatiche.
Esempio

register int variabile;

L'ultima classe di memorizzazione e' costituita dalle variabili dichiarate come extern.
Una variabile dichiarata internamente ad una funzione, come abbiamo visto precedentemente, viene considerata locale.
E' possibile renderla visibile all'esterno della stessa mediante una dichiarazione extern.

extern char pippo[];

A questo punto che abbiamo visto quello che sono le variabili e il significato delle funzioni possiamo includere nei nostri discorsi il punto chiave della programmazione Object Oriented ovvero quello relativo alla creazione delle CLASSI.
Volendo fare un discorso metaforico potremmo dire che qualsiasi cosa del nostro mondo può essere vista come un oggetto che a sua volta è composto da altri oggetti ciascuno dei quali possiede determinate caratteristiche e modi di funzionamento.
Avevamo detto prima che volevamo ricordarci a cosa si riferivano determinati dati potevamo utilizzare il concetto di struttura per incapsulare tutte le variabili relative ad un determinato oggetto.

Nel Linguaggio C++ il concetto di classe è un ampliamento di quello di struttura ovvero un contenitore idoneo a contenere sia i dati che i metodi di funzionamento relativi a un qualche cosa.
Non ho usato il termine oggetto in quanto verrebbe se no da pensare che quelli definiti con il termine di oggetti siano di fatto le classi del C++.
Ritornando al discorso dell'analisi possiamo dire che ogni problema o che ogni componente del mondo reale può essere scomposto in un certo numero di sottocomponenti.
Anche nell'ambito della programmazione ci ritroviamo a poter utilizzare i concetti di struttura e di classe per descrivere questi componenti.
Facciamo un esempio pratico.
Supponiamo di dover creare un insieme di oggetti idonei a ricevere in input dei dati da tastiera ed a stampare delle stringhe statiche.
Analizzando tutti e due i componenti ci accorgiamo che questi sono disegnati a video e qui rappresentati da dei rettangoli.
Questi rettangoli servono a visualizzare le dimensioni di tali oggetti nell'ambito dell'interfaccia utente.
Quindi indipendentemente dal fatto che poi dentro a questi rettangoli si scriva o si leggano i tasti digitati, i metodi per il disegno dei bordo sarà comune.
Ora prendiamo in esame un sistema che gestisca il disegno di rettangoli a video.
Come proprietà memorizzabili in variabili ci saranno quelle delle posizioni di x e y e della larghezza e altezza.
A livello di metodi funzionali invece ci sarà una funzione che sfruttando questi dati esegue il disegno del rettangolo.
Volendo incapsulare tutti gli appartenenti a questo blocco potremmo usare la definizione di classe con :

```
class rectangle {
        int x, y;
        int width, height ;
};
```

Volendo inserire anche il metodo di disegno la classe si tramuterebbe in :

```
class rectangle {
        int x, y;
        int width, height ;
        void designRect();
};
```

Ora la classe rettangolo può essere usata per creare un oggetto rettangolo il quale possiede tutto quello che gli necessita per il suo ciclo vitale e funzionale, semplicemente dichiarando delle variabili con questo tipo.
Potremmo quindi fare :

```
class rectangle rect1;
```

Quindi potremo prima assegnare le variabili con :

```
rect1.x = 23;
…
rect1.designRect();
```

E' inutile dire che sicuramente la funzione designRect() utilizzerà i suoi dati interni alla classe per l'esecuzione del disegno del rettangolo.
Ora se dovessimo continuare la creazione degli oggetti di cui avevamo parlato potremmo fare una cosa.
Una delle caratteristiche della programmazione object oriented è legata al concetto di ereditarietà.
In pratica creando altre classi si potrebbe usare questa appena definita per fare nascere la nuova classe.
Sicuramente sia la classe d'input che quella per la gestione della stringa statica a video dovrà utilizzare una funzione per disegnare il rettangolo che delimita il bordo del campo.
Invece di scrivere internamente tale funzione potremo creare la nuova classe in modo che erediti tale funzionalità dalla classe rectangle.

```
class   editField : public rectangle
{
        char valore[256];
        void leggiValore();
};
```

Allo stesso modo potremo definire la classe :

```
class   staticField : public rectangle
{
        char stringa[256];
        staticField(char *value);
        void printValue();
};
```

Ereditando dalla classe rectangle la fnzione per il disegno del bordo si potrà fare :

```
editField       field1;

field1.x = 23;
…
field.designRect();
```

In questo caso ho usato la funzione ereditata per farvi vedere che ereditandola la si può utilizzare ma sicuramente questa verrebbe usata da qualche funzione interna alla classe che disegnerà il bordo al momento della creazione dell'oggetto editField.
Questo è un esempio molto generico in quanto poi vedremo che dentro alle classi esistono degli attributi che permetterebbero di definire come privati certi membri, pubblici altri ecc.
Quello che mi interessava era quello di far capire come è possibile sfruttare il concetto di ereditarietà ovvero quando un classe creata a partire da un'altra eredita tutte le funzioni incluse in questa o in quelle da cui a sua volta quest'ultima classe deriva.
Nella teoria delle classi esistono due concetti particolari ovvero quelli del costruttore e quello del distruttore.
In pratica questi due sono due metodi che vengono richiamati, se implementati nella classe, all'atto della creazione della classe e all'atto della sua distruzione ovvero nell'istante in cui viene richiesta la disallocazione.
Avevamo visto prima che grazie al concetto di puntatore era possibile utilizzare l'allocazione dinamica ovvero quel tipo in cui si richiede al sistema, tramite una funzione, di allocare una certa quantità di memoria e quindi di assegnare l'indirizzo di partenza al puntatore di destinazione.
La programmazione Object Oriented include un nuovo operatore che permette di creare dinamicamente degli oggetti.
Stiamo parlando dell'operatore new.
Invece di richiedere l'allocazione statica di una classe, ad esempio con :

```
className       varName;
```

Potremmo fare :

```
className       *varName = new className;
```

Il sistema richiederebbe al sistema operativo di allocare da qualche parte della memoria di grandezza sufficiente e assegnerebbe l'indirizzo al puntatore.
All'atto della creazione, se gestito all'interno della classe, verrebbe richiamato il costruttore della classe che a livello di dichiarazione in genere è come un metodo che possiede lo stesso nome della classe.
Ad esempio :

```
class   className {
        className();
        ~className();
};
```

className() è il metodo costruttore, che come abbiamo apena detto verrebbe richiamato in fase di creazione della classe.
Il secondo metodo invece, quello preceduto da ~ è il distruttore ovvero il metodo richiamato in fase di distruzione della classe.
La creazione dinamicamente si ottiene con **new**.
La distruzione di una classe avviene quando non ha più scopo oppure quando viene richiesta la distruzione tramite l'operatore **delete**.

```
#include <iostream>
using namespace std;

class Box                // Class definition at global scope
{
   public:
      double length;     // Length of a box in inches
      double breadth;    // Breadth of a box in inches
      double height;     // Height of a box in inches

      // Constructor definition
      Box(double lv, double bv, double hv)
      {
         cout << endl << "Constructor called.";
         length = lv;                    // Set values of
         breadth = bv;                   // data members
         height = hv;
      }

      // Function to calculate the volume of a box
      double Volume()
      {
         return length * breadth * height;
      }
};

int main(void)
{
   Box Box1(78.0,24.0,18.0);   // Declare and initialize Box1
   Box CigarBox(8.0,5.0,1.0);  // Declare and initialize CigarBox
   double volume = 0.0;        // Store the volume of a box here
   volume = Box1.Volume();     // Calculate volume of Box1

   cout << endl
        << "Volume of Box1 = " << volume;
   cout << endl
        << "Volume of CigarBox = "
        << CigarBox.Volume();
   cout << endl;

   return 0;
}
```

Parlavamo prima dicendo che con il compilatore C vengono distribuite delle librerie contenenti un numero molto elevato di funzioni destinate allo svolgimento dei compiti più diversi.
La distribuzione di Windows ha portato a implementare con il compilatore un'altra serie enorme di librerie di funzioni ciascuna delle quali è destinata al controllo di windows stesso, come ad esempio il controllo delle finestre, delle dialog, dei campi di edit ecc.
L'avvento del C++ ha portato alla creazione di classi che incapsulano tutte queste API all'interno di quelle che sono state chiamate con il nome di Microsoft Foundation Class (MFC).
Scrivendo un programma in C normale potremo, al fine di gestire le varie caratteristiche del programma utilizzare la API oppure, usando il C++ implementeremo il tutto sfruttando le classi MFC.
Chiaramente il concetto di classe ha permesso di implementare tutte le funzioni a partire da dei semplici oggetti di base.
Andando a vedere la carta gerarchica delle classi incluse dentro a MFC ci accorgeremmo che tutte derivano dalla classe CObjetc.
Il pericolo di usare il C++ venendo dal C potrebbe essere quello che avevo fatto io ai tempi in cui avevo fatto il passaggio da uno all'altro, verso il 1987.
Quando scrivete un programma con la metodologia object oriented cercate di identificare ogni singolo componente del programma osservandolo bene e cercando di definirlo con tutte le sue variabili e con tutti i suoi metodi necessari a farlo funzionare.
Se l'oggetto lo avrete strutturato bene questo potrebbe essere preso e spostato in un altro ambito continuando a vivere tranquillamente e quindi funzionare.

In pratica dicevo che quando nel 1987 ho comprato la prima versione di compilatore C++, lo Zortech, continuai a scrivere tranquillamente codice in C utilizzando le classi in alcuni punti ma senza che poiu alla fine il programma risultasse un'unica classe.
In altre parole più che scrivere in C++ scrivevo programmi in C orientati al C++.
Un giorno acquistai C++View che altro non era che una trasposizione dello Smaltalk verso il C++.
Questo sistema di sviluppo non ti lasciava uscire dalle classi per cui diventava necessario andare a vedere i singoli componenti di base e mano mano permetteva di creare nuove classi ma sempre partendo da quelle già esistenti.
Alla fine del tutto il programma era una classe che aveva ereditato gerarchicamente tutte le altre classi.
Guardate la tabella di cui vi ho parlato, quella della struttura delle classi MFC, e capirete quello che voglio dire.
D'altra parte, come abbiamo già detto, Windows costituisce l'ambiente idoneo all'incapsulamento di tutte le componenti sia che questi siano relativi alla gestione dell'interfaccia utente, che per quanto riguarda le gestioni delle basi di dati

Le parole chiave per il controllo dei flussi

Fino ad ora abbiamo in parte parlato della strutturazione di un programma, chiaramente facendolo nei limiti offerti da un testo di questo tipo, tralasciando però le parole chiave utilizzate per il controllo del flusso dei programmi.
Se non esistessero costrutti tali per applicare quelli che sono i concetti chiave di tutti i linguaggi quali ad esempio i controlli, i cicli interattivi ecc. un programma verrebbe considerato semplicemente come una sequenza lineare di istruzioni che verrebbero sempre eseguite allo stesso modo.
I sistemi orientati ai controlli dei dati invece permettono di variare i flussi di esecuzione.
La parola chiave fondamentale è quella che permette di creare il costrutto :

if(condizione) { ….. } [else { ….. }]

La parte tra parentesi quadre è opzionale.
La condizione specificata all'interno delle parentesi rotonde utilizza gli operatori relazionali per verificare i valori all'interno di variabili o quelli restituiti da funzioni.
Gli operatori relazionali sono :

Operatore	Nome
!	NOT o negazione
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
%	Modulo
~	Complemento ad uno
<<	Shift a sinistra
>>	Shift a destra
<	Minore di
>	Maggiore di
<=	Minore uguale di
>=	Maggiore uguale di
==	Uguaglianza
!=	Diverso
&	AND su bit, indirizzo di
|	OR su bit
&&	AND logico
||	OR logico
++	Incremento
--	Decremento
=	Assegnazione

Fate attenzione di una cosa.
Per confrontare due variabili e vedere se queste sono uguali si deve utilizzare l'operatore relazionale

==

che è diverso dall'operatore di assegnazione =
Esistono controlli ridondanti del tipo :

if(a) ….

Questo controllo starebbe a dire : SE A E' VERO
Una variabile è sempre vera se contiene un valore diverso da 0.
L'errore più comune di inizia con il C è quello di creare costrutti del tipo :

if(a = 5) ….

Invece di

if(a == 5) …

Nel primo caso cosa capiterebbe ?
In pratica prima di eseguire la valutazione di a avverrebbe l'assegnazione di a = 5.
Essendo a = 5 la valutazione darebbe vero in quanto, come abbiamo detto, qualsiasi valore differente da 0 è VERO (TRUE).
Mediante i congiuntivi e i disgiuntivi AND e OR si possono creare costrutti valutativi formati da piu' valutazioni.
In altre parole si potrebbe fare :

if(a == 5 && b != 6) …. // Se a è uguale a 5 E b è diverso da 6 ….

Gli operatori sono :

AND &&
OR ||

Attenzione che esistono anche gli operatori logici a livello di istruzioni matematiche logiche e precisamente l'operazione di AND logico e di OR logico che sono rappresentati da uno solo dei simboli di prima.
In altre parole :

AND &
OR |

Ad esempio è possibile fare :

```
a = 14 & 2;
b = b | 16;
```

Comuqnue, tornando alla valutazione del if possiamo aggiungere che è possibile anche valutare I valori di ritorno di funzioni senza fare passaggi tramite assegnazioni.
In pratica dovendo valutare il valore restituito da una funzione potremmo fare :

```
int a = funzione();
if(a == 5) ….
```

Oppure direttamente :

if(funzione()) ….

Il costrutto else sarebbe come dire

```
If(condizione_è_vera) {
        Istruzione 1;
        …..
        Istruzione n;
} else {
        Istruzione 1;
        …..
```

```
        Istruzione n;
}
```

La parte racchiusa nell'else viene eseguita nel caso in cui la condizione risulti falsa.
Nel caso in cui la conseguenza della valutazione fosse di una sola istruzione è possibile anche omettere le parentesi graffe che racchiudono le istruzioni.
Ad esempio sono costrutti validi :

```
if(a == 5)
        funzione1();
else
        funzione2();
```

Oppure :

```
if(a == 67 && b == a)
        funzione();
```

Esistono casi in cui è necessario fare vautazioni multiple.
In questo caso è possibile mettere consecutivamente più if oppure è possibile usare il costrutto

```
switch(valore_da_controllare) {
case    xxx:
        ….
        break;
case    yyy:
        ….
        break;
default:
        ….
        break;
}
```

I vari case corrispondono ai valori mentre il default significa che se nessuno degli altri casi è stato valutato come vero, vengono eseguite le istruzioni interne al default.
Il break dice di interrompere l'esecuzione.
Fate attenzione che se fosse omesso il break l'esecuzione continuerebbe anche con le istruzioni dopo anche se queste fossero parte di un altro case.
Ad esempio :

```
#include <stdio.h>

void main(void)
{
    int sel;
    puts("1 .. Directory");
    puts("2 .. Versione DOS");
    puts("Scelta : ");
    sel = getch();
    switch(sel)
    {
         case 1:
              system("dir");
              break;
         case 2:
              system("ver");
              break;
         default:
              puts("\n\nScelta errata");
              exit(0);
              break;
    }
}
```

Ora vediamo un altro costrutto che permette di creare dei cicli interattivi ovvero di delimitare delle istruzioni in modo che vengano eseguite un certo numero di volte.
Il numero di volte verrà stabilito da valutazioni che verranno fatte dai costrutti.
Il primo è il for()
La sintassi è :

for(init_var;controllo_var;operazione_var) { …. }

Il for è composto da tre parti opzionali interne ovvero una parte relativa all'inizializzazione delle varibili usate per la ripetizione del ciclo, una parte legata al controllo e una dedicata all'operazione.
Ad esempio se sapessimo a priori che un certo gruppo di istruzioni deve essere ripetuto un certo numero di volte potremmo fare :

```
int     indice;

for(indice = 1; indice != 10;indice++) { …. }
```

Il costrutto appena visto sarebbe come dire :

per indice che parte da uno fino a quando indice è diverso da 10 incrementa indice di uno RIPETI

In pratica le parti OPZIONALI (nessuna è necessaria) sono :

ASSEGNAZIONE
CONTROLLO
OPERAZIONE

Se volessimo eseguire all'infinito un certo numero di istruzioni potremmo anche fare :

```
for(;;) {
        ….
}
```

Il ciclo in questo caso verrebbe interrotto da qualche cosa interno al ciclo stesso, o da un return dalla funzione che include il for() oppure grazie ad un istruzione di BREAK che interrompe il for().
Fate attenzione che se sbagliate il controllo potreste anche non uscire più dal ciclo.
Prendete l'esempio :

for(indice = 1; indice != 3;indice+=4) { …. }

In questo caso la prima volta indice varrebbe 1 ma il ciclo subito dopo, visto che l'incremento è di 4 verrebbe subito 5 per cui non sarebbe mai falsa la condizione FINCHE INDICE E' DIVERSO DA 3.
Dicevamo che il for() è costituito da tre parti.
Avremmo potuto eseguire il tutto con un'altra forma di interazione offerta dal C ovvero mediante il while.
Questo ha la seguente sintassi :

```
while(condizione) {
        …
}
```

In pratica dice : ripeti fino a quando la condizione è vera.
La valutazione della condizione segue la sintassi del IF.
Ad esempio è possibile ripetere all'infinito, lasciando il compito di terminare il tutto qa qualche istruzione interna al while, mediante :

```
while(1) {
        ….
}
```

Questo vuole dire :

RIPETI FINO A QUANDO 1 E' VERO

Uno è sempre vero !!!
Chiaramente con il while la valutazione avviene subito all'inizio per cui nessuno ci dice che quelle istruzioni dentro al while verrebbero eseguite almeno una volta.
Se si vuole che queste almeno per la prima volta vengano eseguite, è possibile usare il costrutto :

```
do {
   ….
} while(condizione);
```

In questo cas la valutazione avviene dopo la prima volta che le istruzioni interne vengono eseguite.
Ricordatevi che molte funzioni restituiscono valori che possono essere usati per la valutazione degli if, while, do while, for.
Prendete ad esempio la funzione che compara due stringhe, ovvero strcmp()
Questa eseguela sottrazione della prima stringa passata cme argomento alla seconda per cui il valore restituito è :

< 0 se la prima stringa è minore delle seconda
0 se le due stringhe sono uguali
>0 se la seconda stringa è minore della prima

In questo caso sarebbe possibile fare :

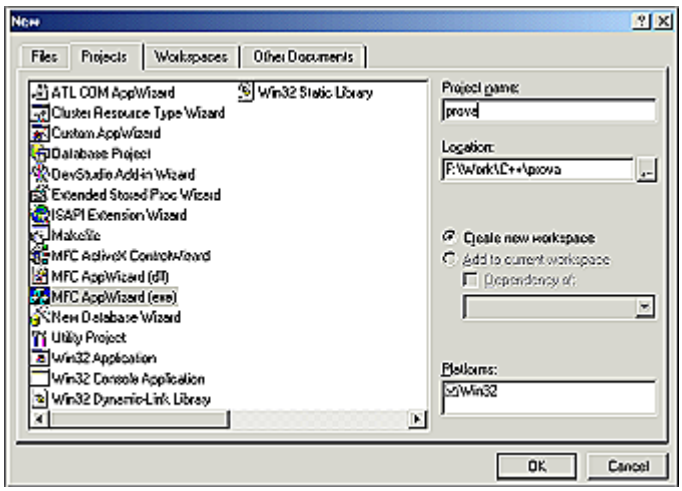
```
char    a[] = "Ciao";
char    b[] = "Flavio";

if(!strcmp(a,b)) ….
```

Oppure :

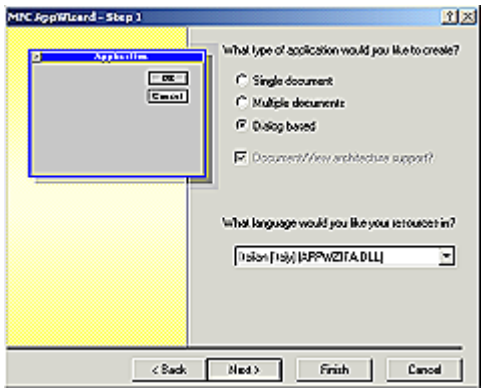
```
while(strcmp(a,b)) ….
```

In linea di massima i concetti di Windows

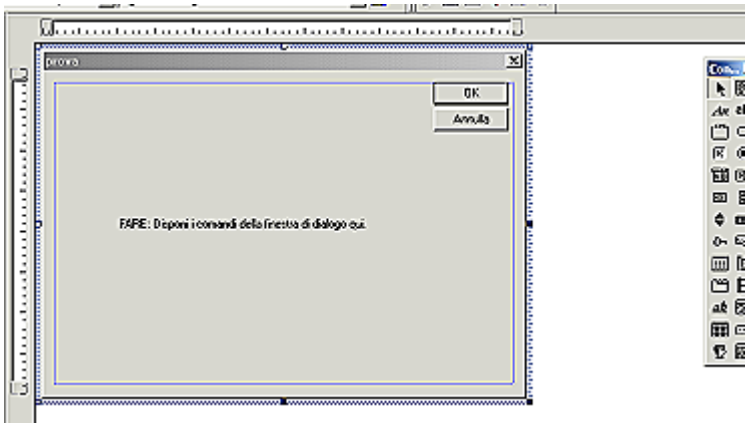
Sotto DOS esistevano diverse funzioni che permettevano di eseguire l'input/output sull'interfaccia utente.
Chiaramente queste funzionalità sono importantissime in quanto all'interno di un programma la maggior parte del codice è quasi sempre legato alla gestione dell'input da parte dell'utente ed in particolar modo al controllo d quanto digitato e successivamente, dopo l'elaborazione dei dati, al suo output sulle maschere video.
La filosofia Object Oriented ha costituito un ambiente ottimale per creare tutti gli oggetti che possono permettere questa interazione con l'utente in ambiente Windows.
Gran parte del lavoro viene svolto, fortunatamente, dai vari Wizard disponbili con i vari sistemi di sviluppo.
Spesso la strutturazione dei programmi, in questo ambiente, è abbastanza complessa in particolar modo per quanto riguarda ad esempio il modello offerto dalle classi MFC distribuite con visual C++.
Dicevamo che fortunatamente spesso dobbiamo solo limitarci a posizionare degli oggetti selezionati da menu o toolbar sopra le dialog create automaticamente dal sistema di sviluppo.
Una volta posizionati questo oggetti, ci saranno delle variabili che li identificheranno e queste verranno usate per accedere ai vari metodi inclusi dentro alle classi da cui questi derivano, che ci permetteranno di manipolare i vari contenuti.
Facciamo un esempio pratico utilizzando il Visual C++.
Attiviamolo e richiediamo, mediante scelta da menu, di creare un nuovo progetto di tipo MFC dandogli come nome prova.



Proseguendo selezioniamo un progetto di tipo DIALOG BASED.
Visual C++ gestisce tre tipi fondamentali di progetti ciascuno dei quali tratta in un certo modo l'interfaccia utente.
Chiaramente per l'esempio ho scelto quello più semplice in quanto di fatto non è pretesa di questo testo essere una guida approfondita di programmazione in ambiente Windows.

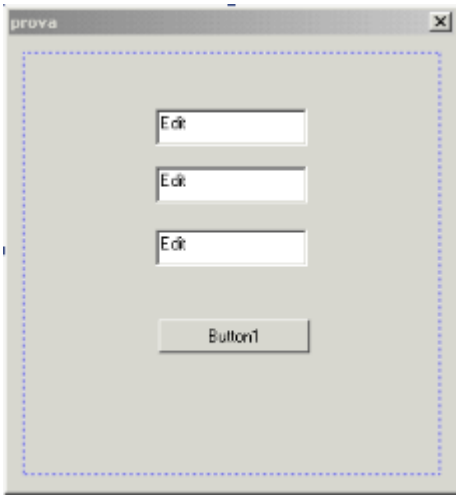


Lasciamo gli altri parametri di default fino a quando Visual Studio creerà la dialog di lavoro a video.

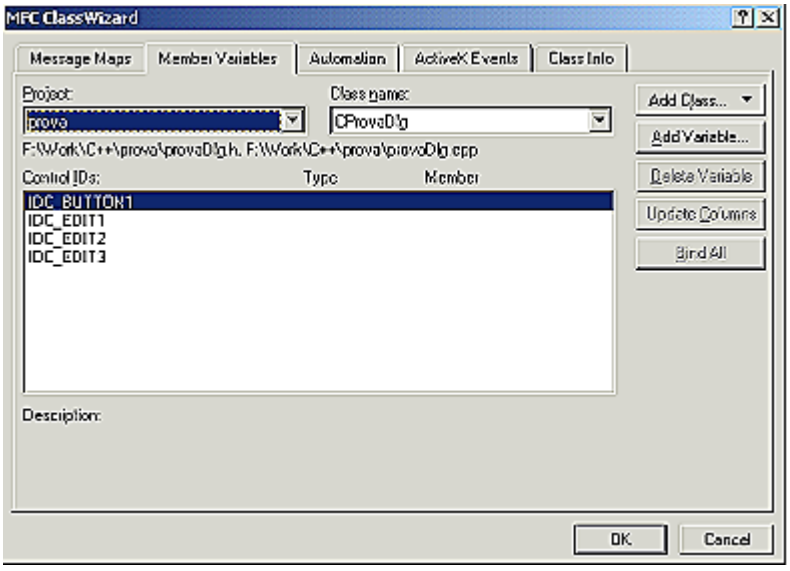


Windows è sempre stato, si dai primi tempi, EVENT BASED ovvero basato agli eventi o ai messaggi.
In pratica il sistema intercetta qualsiasi evento identificando l'oggetto a cui è destinato.
Chiaramente il codice scritto dall'utente può intercettare e trattare tale messaggio o ignorarlo.
Prendiamo ad esempio un pulsante.
Premendolo verrà generato un evento destinato a questo, intercettando il quale sarà possibile creare del codice in funzione di questo.
L'implementazione delle classi MFC ha permesso di incapsulare anche i messaggi facendoli diventare una proprietà intercettabile degli oggetti stessi.

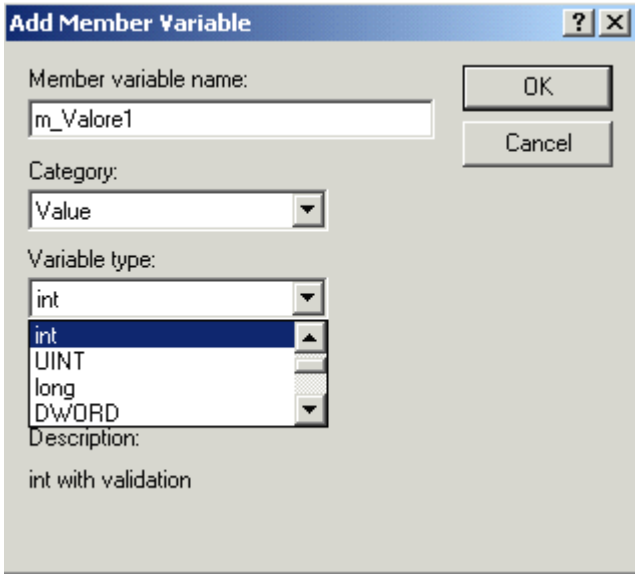
La dialog di default viene creata con tre oggetti di default sopra ovvero un campo statico con un scritta e due pulsanti.
Ora cancelliamo tutti questi tre oggetti da video.
Supponiamo di voler creare un programma che inserendo due valori in due campi di edit, dopo aver premuto un pulsante, visualizzi il risultato in un terzo campo.
Inseriamo tre campi di edit :



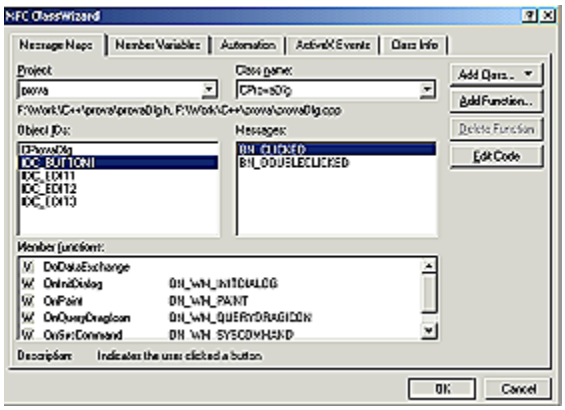
Ora dovremo identificare i tre campi in modo tale che sfruttando i vari metodi a disposizione per questi tipi di oggetti sia possibile leggere e settare i valori.
Posizioniamoci sul primo e dopo aver premuto il tasto destro del mouse scegliamo il CLASS WIZARD.
Il class wizard ci permette di fare due cose e precisamente di attribuire dei nomi agli oggetti e di intercettare gli eventi su di questi.
Per creare tre variabili che identificheranno i tre campi selezioniamo il tabulatore Member Variables.
I questo ci verranno mostrati gli oggetti a video.
Selezionandoli e scegliendo Nuova Variabile potremo selezionare un nome identificativo e un tipo.



Scegliamo IDC_EDIT1, premiamo Add Variable e damogli un nome e selezioniamo come tipo INT



Ripetiamo cambiando nome per tutti i tre i campi di edit.
In pratica nel nostro programma avremo tre variabili m_Valore1, m_Valore2 e m_Valore3 che avranno tutte le proprietà dei campi Cedit.
Andate a vedere le proprietà dei campi Cedit e vedrete quali sono i metodi che permettono di gestirle.
Ora dovremo invece intercettare il messaggio creato premendo il pulsante.
Seezioniamo il pulasante e dopo aver pigiato il tasto destro del mpuse selezioniamo Class Wizard.
Questa volta però selezioneremo il tabulatore Message Map.
In base ai tipi di messaggi possibili per il tipo di oggetto selezionato ci verranno mostrate le possibilità di scelta.
Nel caso di pulsanti della classe Cbutton i messaggi possibili sono BN_CLICKED e BN:DOUBLECLICKED.
Selezioniamo il primo e premiamo Add Function.



Dopo la creazione nela lista in basso comparirà la fnzione relativa alla gestione del messaggio.
Selezioniamo direttamente edit code in modo da entare subito nell'edit della funzione.

```
void CProvaDlg::OnButton1()
{
	// TODO: Add your control notification handler code here

}
```

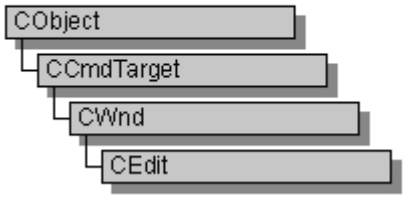
Posizioniamocie dentro e scriviamo :

```
void CProvaDlg::OnButton1()
{
	UpdateData(TRUE);
	m_Valore3 =  m_Valore2 + m_Valore3;
```

```
        UpdateData(FALSE);
}
```

La prima e l'ultima funzione UpdateData() servono ad aggiornare i campi.
Con parametro TRUE legge i valori dalla dialog e li assegna alle variabili, mentre con parametro FALSE dfa il contrario.
In questo caso il programma è terminato.
Basterà compilarlo e provarlo.
L'esempio era solo orientato a farvi vedere come è possibile creare variabili che rappresentano oggetti di una certa classe e come ciascuna di queste possiede attributi e metodi atti a gestirle.
Ricordatevi sempre che le classi sono tante e i metodi sono un infinità per cui è sempre necessario avere l'help sottomano.
Andando a vedere per ogni tipo di classe vedrete la gerarchia, ad esempio quella che segue è relativa ai campi di tipo Cedit.

CEdit



Ora vedremo a caso alcune classi particolari dalle quali si potrà comprendere la logica di funzionamento del Visual C++.
Sicuramente il discorso della programmazione in Visual C++ non è una cosa che è possibile apprendere soltanto leggendo le poche pagine di questo testo, ma sicuramente vedendo come vengono gestiti certi tipi di oggetti risulterà più semplice comprendere la logica che poi alla fine è alla base di tutti gli oggetti inclusi in un linguaggio Visual.
Ve lo ripeto nuovamente.
Quando scrivete un programma pensate passo a passo tutto quello che deve fare questo e cercate nell'ambito degli oggetti a disposizione quelli che dispongono delle caratteristiche migliori per riuscire a risolvere certi tipi di problemi.
Quando tenevo corsi di programmazione come primi algoritmi dicevo agli alunni di gestire quello relativo al compito di alzarsi da dove si è seduti e di andare a comprare le sigarette.
Questo problema eseguito a step si risolverebbe in un serie di azioni e di valutazioni mediante le quali si esegue la scelta delle istruzioni successive.
Ad esempio potremmo dire :

Ci alziamo
Iniziamo a camminare
Arrivati alla porta guardiamo se questa è aperta
Se è chiusa la apriamo
Se è già aperta proseguiamo a camminare
….

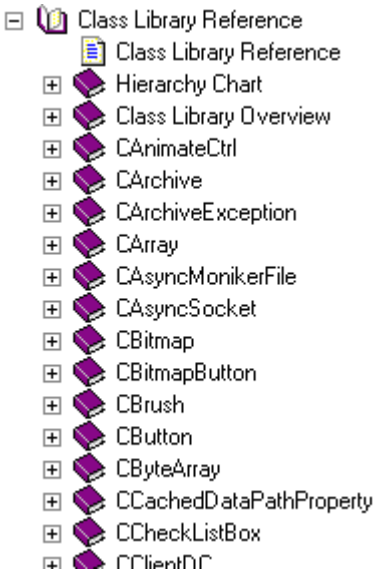
lo programma è la stessa cosa.
Pensate a cosa dovete fare partendo dal presupposto che logicamente il tutto lo possiamo già suddividere in alcuni tipologie di azioni come ad esempio :

OPERAZIONI LEGATE ALLA GESTIONE DELL'INPUT/OUTPUT DALL'INTERFACCIA UTENTE
OPERAZIONI DI MEMORIZZAZIONE SU FILES DEI DATI
OPERAZIONI INTERNE DI CALCOLO
OPERAZIONI DI GESTIONE PERIFERICHE (STAMPANTI, MODEM, RETI)

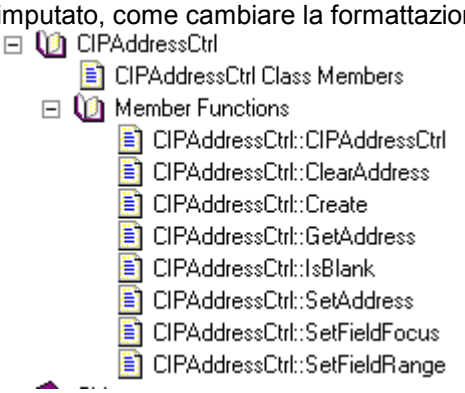
Pensate che gran parte del programma è sempre legato alla gestione del dialogo con l'utente.
Il controllo dei dati inseriti occupa gran parte degli algoritmi tanto che se il programma fosse di fatto per uso personale potremmo ridurre notevolmente il codice eliminando quello relativo ai controlli, agli abbellimenti estetici, all'adattamento dei dati ecc.
Ad esempio se un istruzione di memorizzazione in database di una data pretendesse il formato :

#12/27/2001#

per uso personale potremmo scrivere direttamente la data in quel modo cosa che se invece il programma è destinato a qualche altra persona sarebbe impensabile pretendere che questa scrivesse in questo modo la data richiesta.
Tutto questo per dire che, come ho appena detto, l'help indirizzato alla ricerca delle funzionalità è una cosa fondamentale.



Dovete inserire un valore di un IP ?
Cercate nelle classi e troverete la classe CIPAddressCtrl che permette di gestire l'input nel formato giusto.
Una volta trovata la classe guardate i metodi che contiene per vedere come è possibile leggere il valore imputato, come cambiare la formattazione, ecc.



Nell'immagine di prima vediamo appunto i metodi della classe portata ad esempio.
Se ad esempio volessimo gestire su una nostra dialog un campo di questo tipo, lo potremmo posizionare visivamente, dichiarare una variabile derivata da questa classe mediante il Class Wizard e poi usare tramite il nome da noi assegnato tutti metodi che ci interessa usare per svolgere certi compiti.
Supponendo che abbiamo chiamato ipValue la variabile della classe CIPAddressCtrl, potremmo sapere se il valore è nullo tramite la chiamata a :

if(ipValue.IsBlank()) ….

Riporterò nuovamente la metodologia di creazione di un programma sperando che serva a chiarire il concetto da cui parte la creazione di un programma.

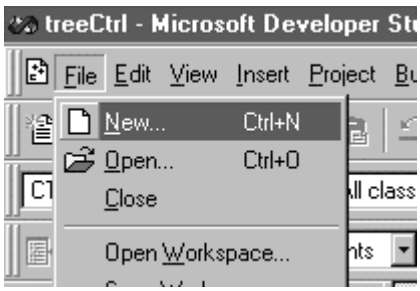
La creazione di un programma

Esistono alcune argomentazioni le cui caratteristiche vengono riportate su volumi che non di rado superano le 800 pagine.
Prendiamo ad esempio il manuale di Microsoft Jet che possiede 800 pagine pulite.

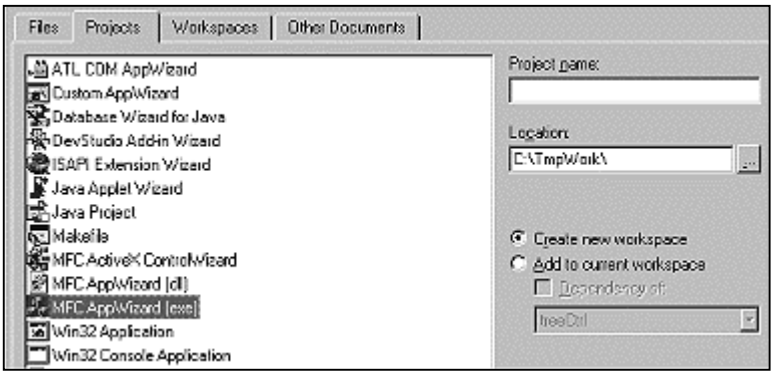
Alcune volte tutti i nozionismi riportati su questi risultano esagerati per gli scopi che ci si prefigge di raggiungere e comunque un inizio piu' dolce a volte riesce ad evitare la confusione che spesso nasce scontrandosi all' inizio con documentazioni ciclopiche.
Come dicevo spesso un piccolo esempio funzionante costituisce una buona base su cui poi aggiungere tutti gli altri nozionismi.
Nel caso del Visual C++ esiste il Developer Studio con il suo Class Wizard che permette di raggiungere immediatamente dei risultati.
Senza contare che a volte quello che si riesce fare con questo e' piu' che sufficiente per quello che si vuole fare.
Il seguente manualetto non contiene un trattato di teoria pura ma, argomento dopo argomento, mostra come affrontare certe problematiche con il class wizard.
Tutti gli esempi sono semplicissimi e vengono mostrati passo a passo anche con l' aiuto di immagini.
Non ci si prefigge traguardi complicati ma si vuole solo mostrare le tecniche per eseguire certe funzioni.
Se ad esempio l' argomento sono le finestre suddivise magari le dialog che verranno create per costituire le viste non conterranno neppure campi.
Una volta imparata la tecnica i campi da inserire li selezionerete voi.
Normalmente sono sempre stato ostile a certi volumi che seguivano esageratamente passo a passo la creazione di un programma preferendo altri testi che curavano maggiormente l' aspetto teorico fornendo quantita' notevoli di nozionismi entro i quali spesso era complicato raccapezzarsi.
Anche gli altri volumi che avevo scritto erano indirizzati a persone che conoscevano la materia.
In questo caso ho voluto dedicare quanto segue a tutte quelle persone che hanno iniziato da poco.
Non e' neppure richiesta una conoscenza molto approfondita del C++ in quanto, come ho gia' detto, gli esempi sono documentati passo a passo e le linee di codice da scrivere sono veramente pochissime.

PROCEDURA STANDARD DI CREAZIONE

In tutti gli esempi verrà utilizzata la stessa procedura di creazione del prototipo.
Riporto ora la procedura di creazione tramite class wizard e nei capitoli che seguiranno mi riferiro' a questa procedura.
Verranno sottolineate le differenze quando queste si riterranno necessarie.
Il class wizard dispone della possibilità di creare tre tipi di applicazioni e precisamente SDI, MDI e Dialog Based.
Tutti gli esempi saranno SDI o al limite Dialog Based anche se non dovrebbero esserci problemi creandoli di altro tipo.

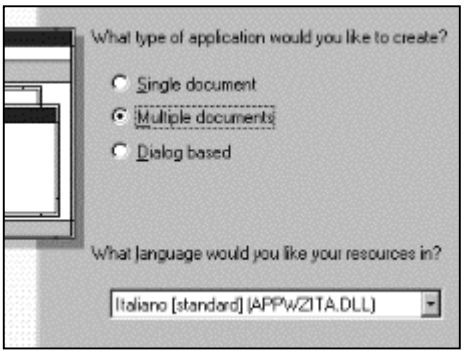


Si inizia selezionando la voce NEW dall' opzione di menu FILE.
Selezionando la linguetta del tabulatore PROJECTS si dovra' selezionare la voce relativa a MFC AppWizard (exe).
Le voci in questa lista di scelta possono essere differenti da applicazione ad applicazione.
La voce appena definita, se si e' fatta richiesta in fase di installazione, dovrebbe essere presente in tutti i sistemi.
Nel caso in cui non sia presente sara' necessario rieseguire l' installazione del Visual C++.
L' installazione potra' essere eseguita parzialmente selezionando nel SETUP CUSTOM la voce relativa agli MFC.

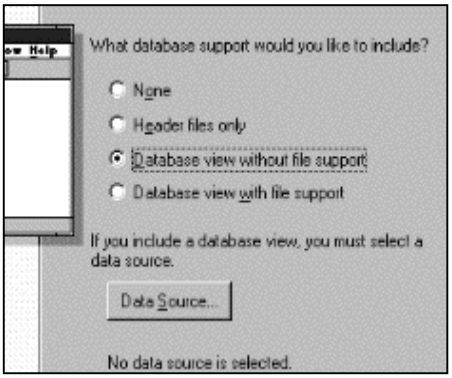


In questa fase sara' necessario assegnare all' applicazione un nome e un path di destinazione.
Immediatamente dopo si dovra' selezionare il tipo di applicazione.
I tipi sono quelli che abbiamo detto prima (SDI ecc.)
Questa scelta influira' anche sul numero di files che verranno generati.
In caso SDI e MDI verranno creati anche i file con il codice relativo alla gestione delle viste e dei documenti.
In pratica l' applicativo si atterra' al modello document/view di Microsoft.
Questa filosofia definisce che esiste una parte del codice che si interessa della gestione dei dati e una parte che invece gestisce il modo con cui questi verranno mostrati.

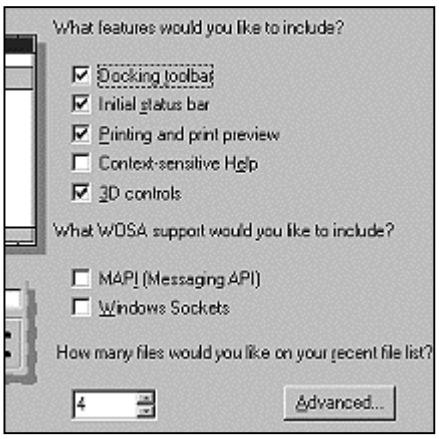
Nel caso di applicazione dialog based avremo un solo modulo di codice destinato alla gestione della dialog.



Nel successivo passaggio viene richiesto sesi desidera utilizzare le funzioni per la gestione di basi di dati DAO.
Puo' essere richiesto se si intende non usare funzioni indirizzate a tali gestioni, se includere solo gli headers con le dichiarazioni, se inserire il codice per una gestione delle viste senza quelle di supporto dei file (Open, Save ecc.) o se inserire tutto.
Nei nostri casi verra' scelto di non utilizzare le funzioni database se non negli esempi dedicati a questo.
Anche in questo caso verra' richiesto solo di includere gli headers.
Nello step che segue verrà invece richiesto che tipologia di documento si intende inserire nel proprio programma.
In tutti gli esempi selezioneremo None.



Nel passo seguente dovremo selezionare quale oggetti includere come ad esempio le toolbar, delle voci di menu ecc.
Nei nostri esempi cercheremo sempre di crearli piu' semplici possibile per cui potremmo deselezionare tutto o a limite lasciare come ci viene proposto e successivamente ignorare il codice superfluo.



Eseguita questa scelta dovremo specificare se richiedere il commento dei file generati e se usare le librerie statiche o dinamiche.
Utilizzeremo le librerie definite come 'Shared DLL'.
Arrivati a questo punto, dopo aver mostrato il nome dei file che verranno generati, il sistema li creera' fisicamente.

GESTIONE DATABASE TRAMITE DAO

Nel 1992 Microsoft rilasciò il primo prodotto integrato per la gestione dei database il quale era composto da un interfaccia utente e da un motore per la gestione dei dati.
Questo motore prese il nome di Microsoft Jet e venne utilizzato all' interno di svariati prodotti Microsoft quali Word, Visual Basic e Visual C++.
Gli oggetti per l' accesso ai dati ovvero i DAO (Data Access Object) forniscono il mezzo per manipolare i dati.
I DAO sono anche un componente ActiveX.
La versione 3.5 e' stata rilasciata nell' ultimo quadrimestre del 1996 ed e' ora disponibile in tutti i prodotti rilsaciati da Microsoft compreso l' ultimo figlioletto Visual J++.
I servizi offerti da Microsoft Jet sono :

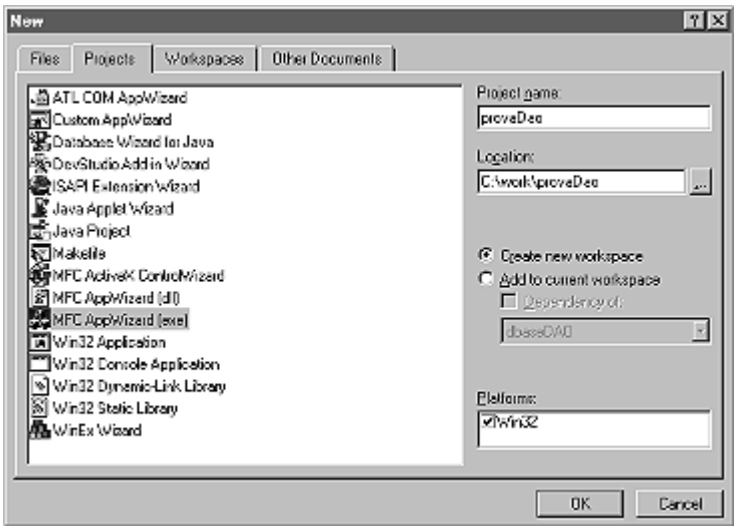
Definizione ed integrita' dei dati.
E' possibile creare strutture atte a contenere informazioni mediante l' uso di database, tabelle, campi, indici, relazioni, query, utenti e gruppi di utenti.
Il controllo dell' integrita' e' indirizzata ad evitare operazioni che possano creare danni nel database.

Archiviazione dei dati .
Esistono funzioni adatte all' immagazinamento delle informazioni tramite il metodo conosciuto come ISAM che nel caso di Microsoft Jet possiede le seguenti caratteristiche.
I record a lunghezza variabile vengono ordinati tramite indici e gestiti in pagine di 2 Kbytes simili a quelle di SQL Server.

Manipolazione dei dati.
Un insieme di metodi adatti alla manipolazione dei dati precedentemente salvati nelle strutture del database.

Reperimento dei dati .

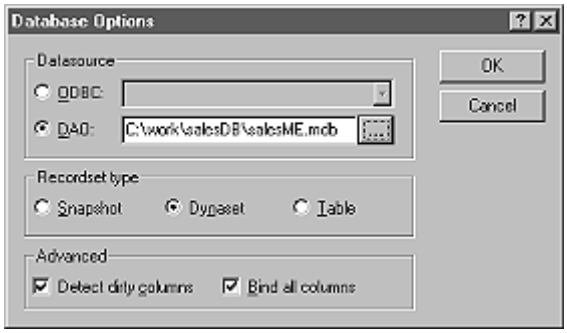
Mediante interrogazioni e funzioni di filtraggio e' permesso reperire informazioni applicando determinate regole di filtraggio.



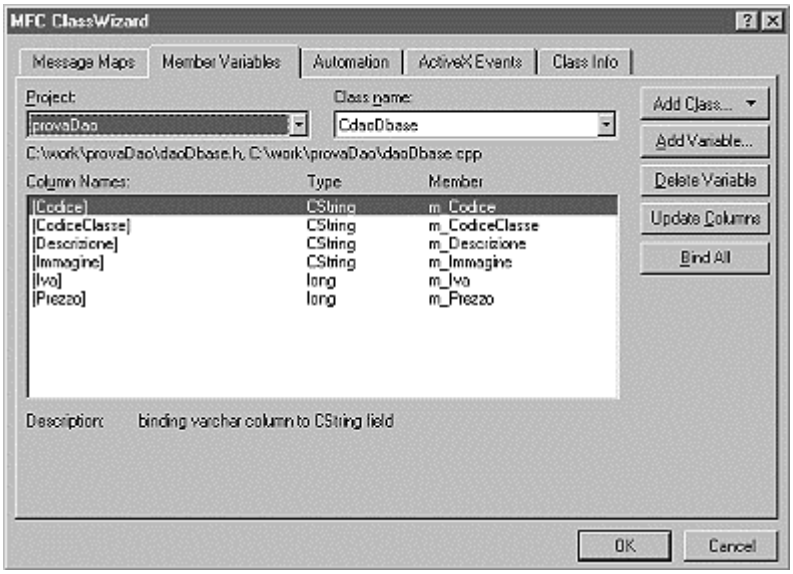
Condivisione
In ambiente multiutente devono esistere metodologie atte a permettere la condivisione dei dati.

Manutenzione del database
Nel caso in cui ci siano problemi o si debba eseguire compattazioni o duplicazioni.;

La versione 3.5 permette inoltre l' esecuzione dei suoi servizi anche tramite ODBC.
Come nel caso del capitolo relativo alle splitter windows creeremo un progetto semplice atto solo a mostrare la tecnica di base da seguire mostrando ogni singolo passo tramite immagini abbinate alla spiegazione.
L' accesso alle basi di dati tramite DAO puo' avvenire utilizzando l' interfaccia ODBC o quella offerta da Microsoft Jet.

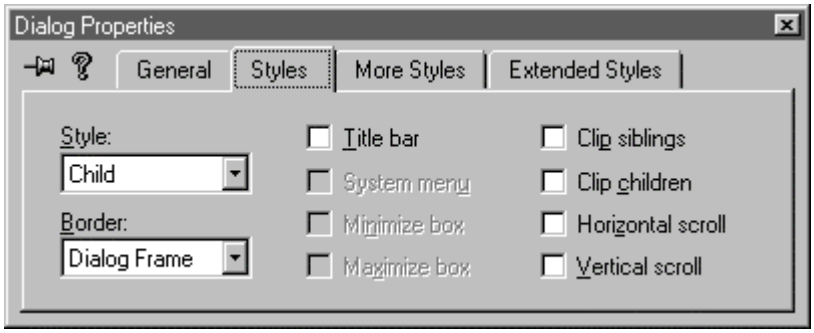


Nel nostro caso utilizzeremo la seconda via.
Iniziamo come al solito creando un nuovo progetto MFC chiamandolo provaDAO, come mostrato nella prima immagine.
Il progetto dovra' essere Single Document con inclusi solo gli headers per la gestione database.
Il resto potrete lasciarlo di default come vi propone il sistema.
Appena generati i files del progetto sara' necessario creare una nuova classe selezionando l' opzione ClassWizard sotto la voce di menu View.
Premete il pulsante Add Class e selezionate New.
Chiamate la classe CdaoDbase e selezionate come classe di base, da cui farla derivare, CdaoRecordset.
Vi verra' mostrata la finestra in cui potrete selezinare se utilizzare ODBC o DAO.
Optate per DAO (vedi immagine precedente) e selezionate il database che vorrete utilizzare.
La finestra vi mostrera' il formato MDB di access anche se di fatto potrete selezionare qualsiasi formato accettato da Microsoft Jet.

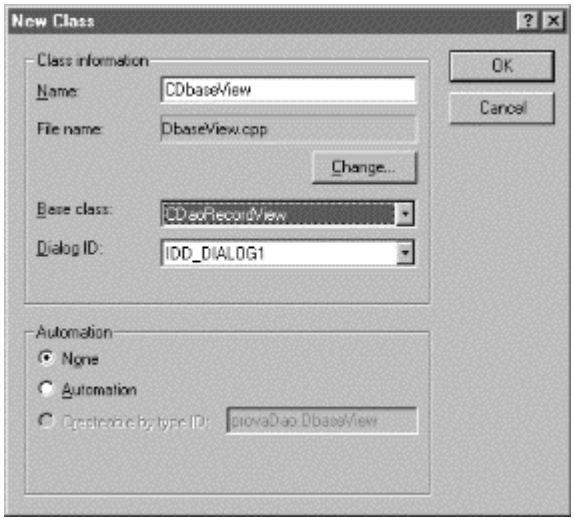


I database riportati sul manuale di Microsoft Jet (Microsoft Jet Database Engine – Guida del programmatore – 800 pagine Microsoft Press by Mondadori Informatica ISBN 88-7131-858-7) sono FoxPro, Dbase Paradox, Excel, Lotus, Ascii, Html.
Nel mio caso mi riferiro' ad un archivio relativo ad un applet destinato alla raccolta ordini in cui l' archivio ha la struttura che potete vedere nel disegno.
Se avete segnato l' opzione Bind all columns alla conferma vi verranno create tutte le variabili relative ai campi.
Tornando alla maschera di creazione della classe selezionate il tabulatore Member Variables e vedrete i nomi dei campi con i relativi tipi.
A questo punto dovremo creare tramite il gestore delle risorse la maschera che verra' utilizzata come vista per la gestione dei dati.
La creazione avviene mediante l' inserimento di una nuova dialog.
La classe di gestione della dialog dovra' essere derivata da CdaoRecordView e dovra' possedere come proprieta' le stesse che vengono utilizzate con quelle derivate da CformView e precisamente :

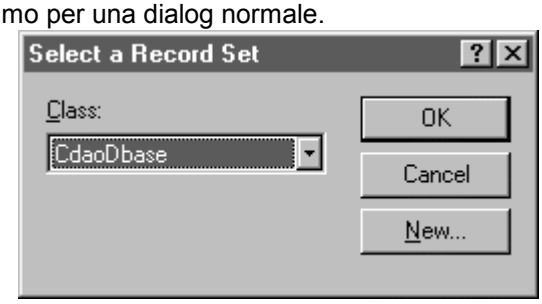
WS_CHILD, WS_BORDER off, WS_VISIBLE off, WS_CAPTION off



Dopo aver settato i parametri come appena detto (vedi immagine) creiamo la nuova classe selezionando il ClassWizard.
Appena avvenuta la selezione ci verra' richiesto se vogliamo creare una nuova classe.
Chiamiamo la nuova classe CdbaseView e selezioniamo come classe di base la CdaoRecordView.
Appena eseguita la conferma ci verra' mostrata la finestra in cui dovremo selezionare la classe CdaoRecordset che vorremo abbinare.



Nel nostro caso avremo solo quella creata precedentemente.
Selezioniamo confermando la classe CdaoDbase e ritorniamo alla creazione della dialog.
A questo punto dovremo inserire i campi relativi all' archivio.
Creaiamo il layout inserendo le labels e i campi di edit come faremmo per una dialog normale.
Per ora ommettiamo i pulsanti che serviranno a navigare sul database e



limitiamoci soltanto ai campi effettivi.
Tutte le funzioni di interscambio dei dati vengono gestite da dalle funzioni DFX_Text che il wizard ha inserito nella funzione DoFieldExchange.
DFX (Data Field Exchange) viene utilizzato dal wizard per la mappatura dei campi.
Nel nostro caso vengono solo utilizzate le funzioni DFX_Text e DFX_Long ma esistono per ogni tipologia di campo anche le altre funzioni.

Data Types and DFX Functions

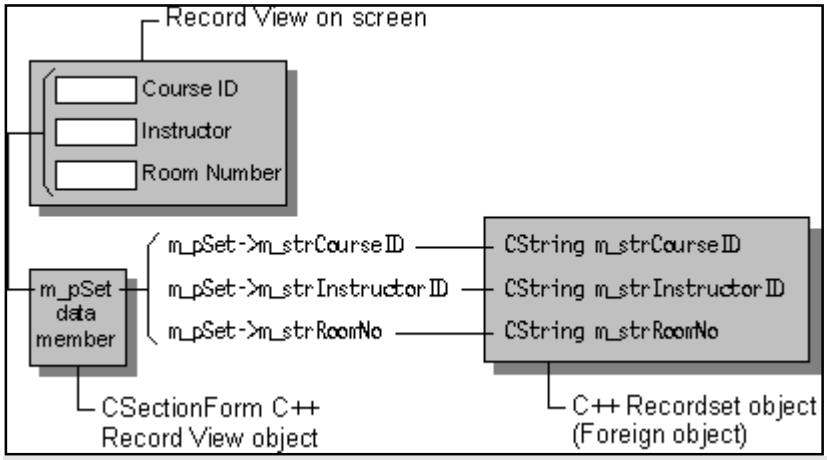
DFX function	C++ data type	DAO data type
DFX_Binary	CByteArray	DAO_BYTES
DFX_Bool	BOOL	DAO_BOOL
DFX_Byte	BYTE	DAO_BYTES
DFX_Currency	COleCurrency	DAO_CURRENCY
DFX_DateTime	COleDateTime	DAO_DATE
DFX_Double	double	DAO_R8
DFX_Long	long	DAO_I4
DFX_LongBinary	CByteArray or CLongBinary*	DAO_BYTES
DFX_Short	short	DAO_I2
DFX_Single	float	DAO_R4
DFX_Text	CString*	DAO_CHAR

La funzione assumera' la seguente forma (questa e' relativa alla struttura dell' archivio che ho utilizzato io per l' esempio).

```
void CdaoDbase::DoFieldExchange(CDaoFieldExchange* pFX)
{
	//{{AFX_FIELD_MAP(CdaoDbase)
```

```
        pFX->SetFieldType(CDaoFieldExchange::outputColumn);
        DFX_Text(pFX, _T("[CodiceClasse]"), m_CodiceClasse);
        DFX_Text(pFX, _T("[Codice]"), m_Codice);
        DFX_Text(pFX, _T("[Descrizione]"), m_Descrizione);
        DFX_Long(pFX, _T("[Prezzo]"), m_Prezzo);
        DFX_Long(pFX, _T("[Iva]"), m_Iva);
        DFX_Text(pFX, _T("[Immagine]"), m_Immagine);
        //}}AFX_FIELD_MAP
}
```

Il Dialog Data Exchange (DDX) permette di mappare i campi di edit presenti su una dialog a quelli di un oggetto Recorset.
Attivando il ClassWizard e selezionando l' ultimo tab relativo a Class Info potrete notare quella definita come Foreign Variable.

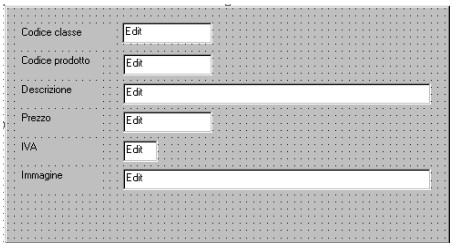


Nel nostro esempio il suo nome e' m_pSet ed e' un puntatore ad un oggetto Recordset.
Gli oggetti relativi ai campi sono identificati da :

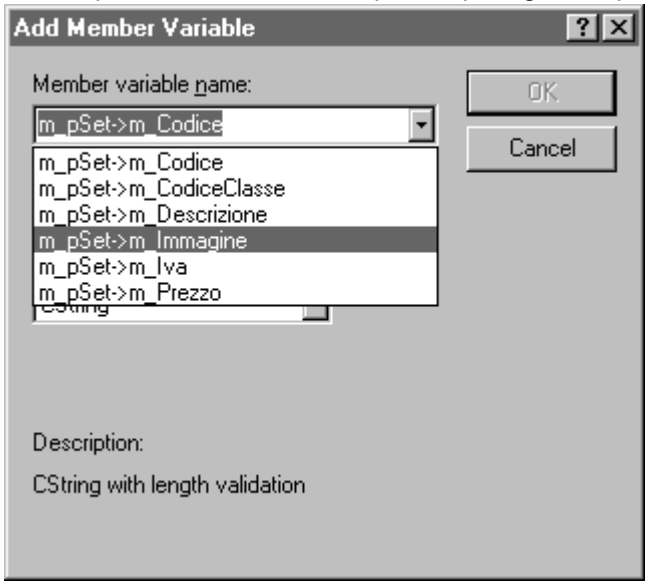
```
m_pSet->nome_campo
```

Ad esempio troveremo tra i nomi creati automaticamente dal Wizard al momento della creazione della classe derivata da CdaoRecordset :

```
m_pSet->m_Codice
m_pSet->m_CodiceClasse
ecc.
```



L' immagine a fianco presa dalla documentazione Microsoft mostra tale relazione.
Preseguiamo ora con la creazione della dialog.
Il layout che ho creato e' il seguente.
Ora si dovranno creare gli abbinamenti tra i campi posizionati sulla dialog e i campi del database.
Riattiviamo il ClassWizard e selezioniamo il tabulatore relativo alla gestione delle variabili.
Per ogni ID dovremo creare una nuova variabile per cui iniziamo a selezionare il primo identificatore relativo al primo campo di edit premiamo il pulsante Add Variable.
Come potete vedere nell' immagine seguente e' possibile selezionare il nome della variabile aprendo la Choice List (Member variable name) e cliccando su uno dei nomi mostrati.
Facendo in questo modo creiamo l' abbinamento tra l' identificatore del campo e la variabile che aveva creato il wizard al momento della creazione della classe CdaoRecordset.
Questo procedimento dovrete ripeterlo per ogni campo.

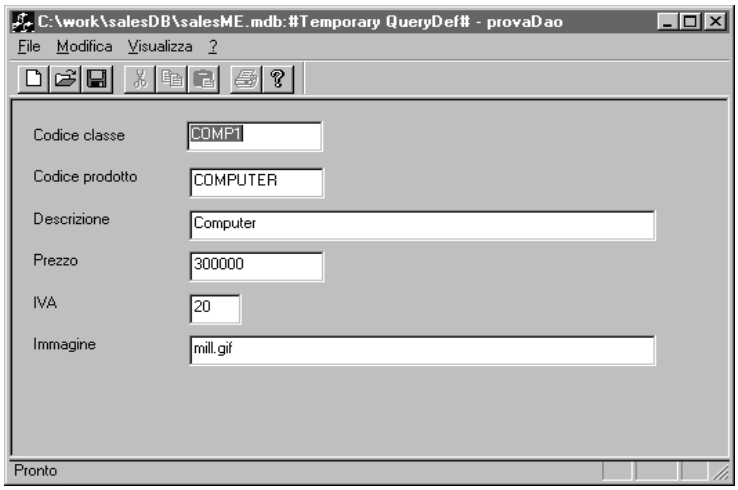


Terminato questo passaggio andiamo a modificare la funzione che registra il template della viewda mostrare in modo tale da far comparire la nostra dialog.
Mediante il ClassWizard andiamo a editare la funzione InitInstance contenuta nel file ProvaDao relativo alla classe CprovaDao.
Nella funzione CsingleDocTemplate sostituiamo CdaoProvaView con CdbaseView.
In pratica la funzione passa da :

```
pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CProvaDaoDoc),
            RUNTIME_CLASS(CMainFrame),
        RUNTIME_CLASS(CProvaDaoView));
```

A

```
pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CProvaDaoDoc),
        RUNTIME_CLASS(CMainFrame),
        RUNTIME_CLASS(CDbaseView));
```



Per fare in modo che venga riconosciuta la classe CdbaseView e' necessario includere il file .H che contiene la dichiarazione.

```
#include "DbaseView.h"
```

A questo punto non sono presenti funzioni che permettano la navigazione ma eseguiamo la prima compilazione per vedere il risultato di quanto fatto fino ad ora.
In pratica verra' mostrato il primo record (sempre che nel database ce ne sia almeno uno).
Infatti dopo la compilazione attivando il programma avremo il seguente risultato.

Ora dovremo, utilizzando i metodi di DaoRecordset, creare le funzioni fondamentali per eseguire la navigazione.
Avevamo detto prima che m_pSet rappresentava il puntatore all' oggetto DaoRecordset che avevamo creato con la classe CdaoDbase.
La classe DaoRecordset contiene le funzioni necessarie ad eseguire lo spostamento sul database, a parte altre funzioni sempre legate alla gestione di quest' ultimo.
Le principali funzioni per la navigazione sono :

```
CDaoRecordset::MovePrev()
CDaoRecordset::MoveNext()
CDaoRecordset::MoveFirst()
CDaoRecordset::MoveLast()
```

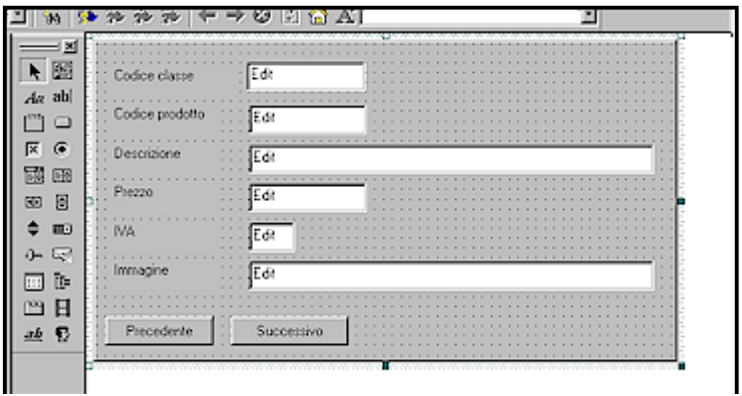
Inoltre sono a disposizione due funzioni che permettono di sapere se e' stata raggiunta la fine o l' inizio del file.
Le funzioni sono

```
CDaoRecordset::IsBOF()
CDaoRecordset::IsEOF()
```

Dovremo, utilizzando queste funzioni, creare dei metodi che verranno abbinati ai pulsanti che svolgono il seguente algoritmo.

PULSANTE VAI AL RECORD PRECEDENTE
Se non e' stata raggiunta l' inizio file muoviti indietro di un record
Se no ritorna senza fare nulla

PULSANTE VAI AL RECORD SUCCESSIVO
Se non e' stata raggiunta la fine del file vai al recoprd successivo
Se no ritorna senza fare nulla.



Editiamo nuovamente la dialog ed aggiungiamo due pulsanti relativi allo spostamento sul precedente e sul prossimo record.
Ora attiviamo il class wizard per creare le due nuove funzioni abbinate ai pulsanti.

Selezioniamo l' ID del primo pulsante e premiamo Add Function chiamando OnPrecedente e OnSuccessivo le due funzioni.
Editiamo il codice ed inseriamo le seguenti righe di programma.
Per la prima funzione il codice sara' :

```
void CDbaseView::OnPrecedente()
{
	if(m_pSet->IsBOF())
		return;
	m_pSet->MovePrev();
	UpdateData(FALSE);
}
```

Per la seconda invece :

```
void CDbaseView::OnSuccessivo()
{
	if(m_pSet->IsEOF( ) )
		return;
	m_pSet->MoveNext();
	UpdateData(FALSE);
}
```

La chiamata a UpdateData(FALSE) mediante il DoDataExchange aggiornera' i dati sulla vista.
E' possibile aggiungere anche gli altri due tasti per posizionarsi sul primo e sull' ultimo record.
Il codice delle due funzioni e' :

```
void CDbaseView::OnPrimo()
{
	m_pSet->MoveFirst();
	UpdateData(FALSE);
}
```

```
void CDbaseView::OnUltimo()
{
	m_pSet->MoveLast();
	UpdateData(FALSE);
}
```

Esistono dei metodi che permettono l' aggiunta di nuovi records e la modifica di quelli esistenti.
Nel caso di aggiunta di nuovi record sara' necessario aggiungere un record vuoto in fondo al file, posizionarsi su questo, eseguire l' edit ed infine richiamare la funzione di update.
Aggiungiamo due nuovi tasti e precisamente Nuovo Record e Modifica.
Per semplificare il tutto faremo in modo che il primo tasto crei un record blank e si posizioni su questo.
Successivamente, dopo aver inserito i dati, potranno essere salvate le modifiche come nel caso di una normale update.

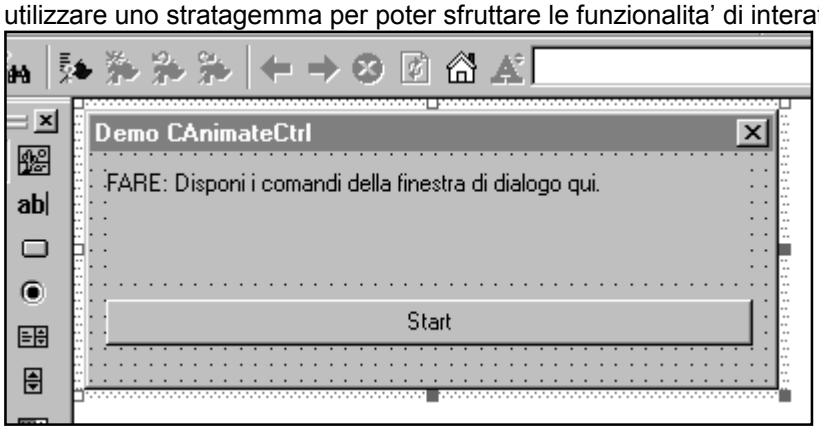
```
void CDbaseView::OnNuovo()
{
	m_pSet->AddNew();
	m_pSet->Update();
	m_pSet->MoveLast();
	UpdateData(FALSE);
}
```

```
void CDbaseView::OnAggiorna()
{
	m_pSet->Edit();
	UpdateData(TRUE);
	m_pSet->Update();
}
```

Comunque a questo punto dopo aver visto come utilizzare il Class Wizard per creare una struttura funzionante sara' possibile utilizzare le svariate funzioni a disposizione.
Ne esistono altre per cancellare i records, per creare dei bookmarks e per settare svariati parametri come ad esempio la dimensione della cache ecc.

CAnimateCtrl

Le CAnimateCtrl sono finestre rettangolari dentro alle quali e' possibile visualizzare una clip AVI.
Prima di iniziare a vedere questo tipo di controllo utilizzate la procedura standard per la creazione dell' applicazione tramite class wizard.
Persemplicita' createla Dialog Based.
Queste classi non possono essere utilizzate direttamente mediante il dialog editor percui e' necessario utilizzare uno stratagemma per poter sfruttare le funzionalita' di interattivita' fornite da questo.



In altre parole durante la fase di editing della dialog posizioneremo nella finestra un altro controllo come ad esempio un Text Label stabilendo in questo modo la posizione e la dimensione che dovra' avere il controllo.
Assegnate l' ID IDC_ANIMFRAME alla label statica utilizzando l' opzione che permette di definire le proprieta'.
In fase di creazione inseriremo anche un pulsante che servira' a far partire il file .AVI.
Nella funzione di InitDialog ricaveremo queste informazioni e le utilizzeremo per sostituire il CAnimateCtrl.

Iniziamo come per gli altri esempi cioe' creando un nuovo progetto utilizzante le MFC.
Questa volta invece di utilizzare un applicazione di tipo SDI o MDI la creeremo Dialog based.
Il file .AVI che utilizzeremo e' un file che si trova in molte applicazioni utilizzato per indicare le funzioni di spostamento dei files.
Il seguente disegno mostra il file AVI nella finestra del lettore multimediale.
Dopo aver generato i files del progetto editate la dialog e dimensinate la label inserita dal wizard con le dimensioni con cui volete visualizzare il file .AVI.
Nel file .h che contiene la dichiarazione della classe relativa alla dialog inseriremo le seguenti dichiarazioni.

```
CAnimateCtrl m_AnimateCtrl;
CRect m_rectAnimateCtrl;
```

La prima variabile e' relativa al controllo stesso.
La variabile Crect invece servira' a ricavare le informazioni relative al posizionamento del controllo che abbiamo inserito come segnaposto.
Richiamate il Class Wizard e editate la funzione InitDialog inserendogli al suo interno il codice che potete vedere qui a seguito.

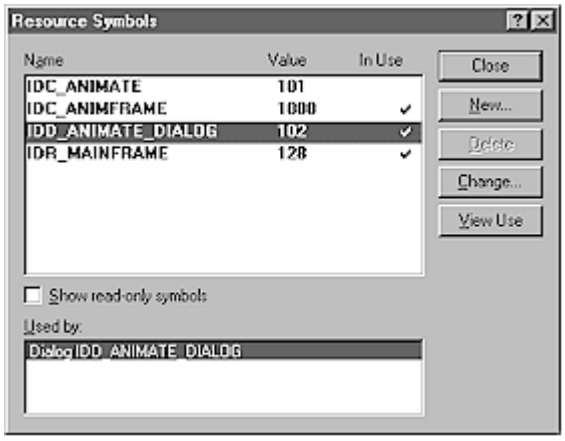
```
BOOL CAnimateDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	CWnd* pFrame = GetDlgItem(IDC_ANIMFRAME);
	pFrame->GetClientRect(&m_rectAnimateCtrl);
	retVal = m_AnimateCtrl.Create(WS_CHILD|WS_VISIBLE|WS_BORDER|ACS_CENTER,
m_rectAnimateCtrl, pFrame, IDC_ANIMATE);

	retVal = m_AnimateCtrl.Open("C:\\WORK\\ANIMATE\\DEBUG\\FILEDISK.AVI");
```

```
        return TRUE;  // return TRUE  unless you set the focus to a control
}
```

Selezionate ora l' opzione View e successivamente Resource Symbols per poter attribuire un ID al controllo creato.
La fase di creazione del controllo e' costituita da due parti .
La prima parte e' relativa alla dichiarazione della variabile, quella fatta nella classe contenuta nel file animateDlg.h
La seconda fase prevede l'uso del metodo Create e del metodo Open che puo' essere utilizzato specificando il path del file .avi oppure l' ID della risorsa.

Il metodo Create assegna un ID al controllo.
Nella maschera Resource Symbol premete il pulsante New e inserite IDC_ANIMATE.
A questo punto dovremo inserire la funzione Play() che permette di far partire l' animazione nel pulsante che avevamo inserito nella dialog.
Mediante il class wizard creiamo una nuova funzione chiamandola Startavi e inseriamogli il seguente codice.

```
void CAnimateDlg::OnStartavi()
{
        m_AnimateCtrl.Play(0,0xFFFF,1);
}
```

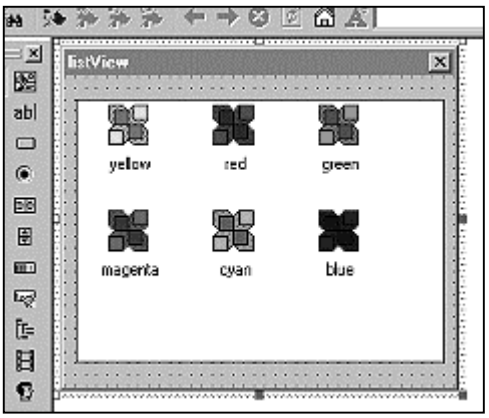


A questo punto possiamo compilare il tutto e vedere il risultato.
La tecnica del controlli animati viene utilizzata spesso per visualizzare azioni come, ad esempio, di pulizia o di copia. Vedi l' operazione di svuotamento del cestino.

CListCtrl e CListView

Anche in questo caso come per i CTreeCtrl e CTreeView esiste una versione indirizzata all' utilizzo tramite dialog editor e una versione indicata in una struttura di tipo document/view.
Questo tipo di controllo comprende una lista composta da un icona e da una label e dispone inoltre di un certo numero di modi legati all' organizzazione della visualizzazione di questi.
I modi sono i seguenti :

```
Icon View
Small icon view
List view
Report view
```



Allo stesso modo dei controlli ad albero e' necessario eseguire una procedura per abbinare delle immagini alla lista stessa.
Prima di vedere il metodo di creazione iniziamo con la creazione di una procedura basata su una dialog utilizzando il metodo standard descritto nell' introduzione.
Appena creata la procedura di base editiamo la dialog ed inseriamogli un controllo ClistCtrl.

Gli attributi visti precedentemente possono essere anche settati mediante il resource editor.
Prima di iniziare a vedere il codice inseriamo nelle risorse le icone che serviranno nella lista.
Nel nostro caso ne ho inserite 4 e gli ho assegnato gli ID che vanno da IDI_ICONLIST1 a IDI_ICONLIST4 costituite da icone che possiedono dimensione di 32 x32 pixels.
Attiviamo il class wizard e come prima cosa creiamo una variabile che rappresenti la nostra lista.
Chiamiamola m_listctrl di tipo CListCtrl .
Editiamo il file .h che contiene la definizione della classe relativa alla dialog ed inseriamogli la dichiarazione di una variabile che rappresentera' la lista delle immagini.

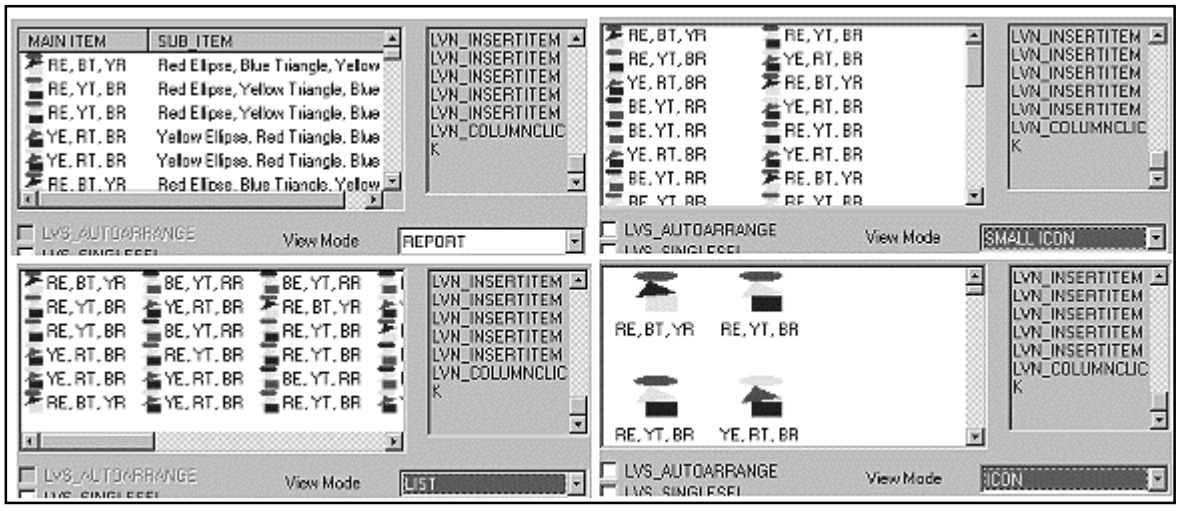
```
CImageList      *m_pimagelist;
```

Sempre mediante il class wizard editiamo la funzione di initdialog ed inseriamo le linee di codice necessarie ad creare la lista immagini.

```
LV_COLUMN   lvcolumn;
LV_ITEM     lvitem;
CCtrldemoApp  *pApp;
CRect   rect;
int i, iIcon, iItem, iSubItem, iActualItem;
TCHAR  rgtsz[2][10] = {_T("PRIMA COLONNA"), _T("SECONDA COLONNA")};
TCHAR  rgtszIconDescrip[LISTICONCOUNT][50], rgtszIconShortDesc[LISTICONCOUNT][12];

_tcscpy(rgtszIconDescrip[0], _T("Descrizione estesa Icona 1"));
_tcscpy(rgtszIconDescrip[1], _T("Descrizione estesa Icona 2"));
_tcscpy(rgtszIconDescrip[2], _T("Descrizione estesa Icona 3"));
_tcscpy(rgtszIconDescrip[3], _T("Descrizione estesa Icona 4"));

_tcscpy(rgtszIconShortDesc[0], _T("ICONA 1"));
_tcscpy(rgtszIconShortDesc[1], _T("ICONA 2"));
_tcscpy(rgtszIconShortDesc[2], _T("ICONA 3"));
_tcscpy(rgtszIconShortDesc[3], _T("ICONA 4"));
// …..
pApp = (CCtrldemoApp *)AfxGetApp();
m_pimagelist = new CImageList();
m_pimagelist->Create(32, 32, TRUE, LISTICONCOUNT, 4);
m_pimagelist->Add(pApp->LoadIcon(IDI_ICONLIST1));
m_pimagelist->Add(pApp->LoadIcon(IDI_ICONLIST2));
m_pimagelist->Add(pApp->LoadIcon(IDI_ICONLIST3));
m_pimagelist->Add(pApp->LoadIcon(IDI_ICONLIST4));
m_listctrl.SetImageList(m_pimagelist, LVSIL_NORMAL);
m_listctrl.GetWindowRect(&rect);

for (i = 0; i < 2; i++)  {
        lvcolumn.mask = LVCF_FMT | LVCF_SUBITEM | LVCF_TEXT | LVCF_WIDTH;
        lvcolumn.fmt = LVCFMT_LEFT;
        lvcolumn.pszText = rgtsz[i];
        lvcolumn.iSubItem = i;
        lvcolumn.cx = rect.Width() * (i + 1) / 3;  // SubItem is twice as large
        m_listctrl.InsertColumn(i, &lvcolumn);  // assumes return value is OK.
}
for (iItem = 0; iItem < 50; iItem++)  // will now insert the items and subitems into the list view.
        for (iSubItem = 0; iSubItem < 2; iSubItem++)
        {
                if (iSubItem == 0)
                iIcon = rand() % 4;
                lvitem.mask = LVIF_TEXT | (iSubItem == 0? LVIF_IMAGE : 0);
                lvitem.iItem = (iSubItem == 0)? iItem : iActualItem;
                lvitem.iSubItem = iSubItem;
                lvitem.pszText = iSubItem == 0? rgtszIconShortDesc[iIcon] : rgtszIconDescrip[iIcon];
                lvitem.iImage = iIcon;
                if (iSubItem == 0)
                        iActualItem = m_listctrl.InsertItem(&lvitem);
                else
```

```
                    m_listctrl.SetItem(&lvitem);
        }
```

Il primo ciclo for serve le colonne alla lista.
Fate attenzione che queste colonne sono necessarie solo se si e' settato il modo REPORT in fase di settaggio delle proprieta' del controllo.
La seguente immagine tratta da un demo Microsoft mostra i quattro metodi di visualizzazione della lista.

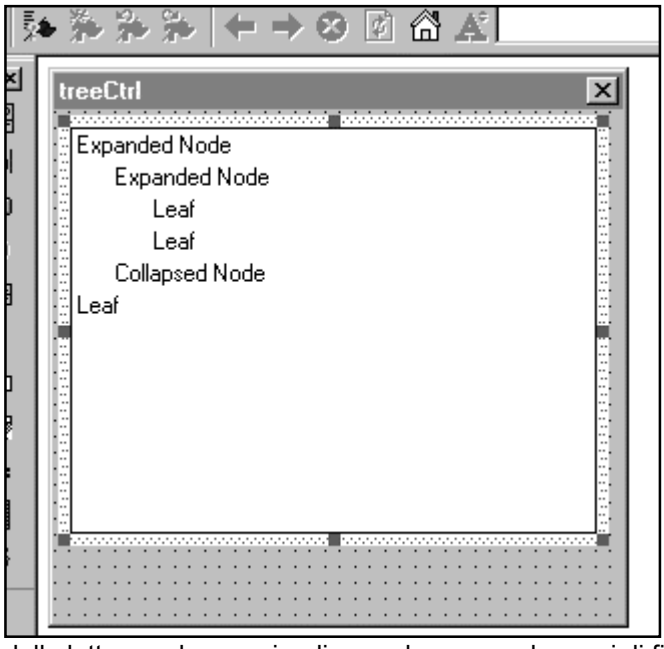


Il secondo ciclo for invece e' quello che crea la lista con 50 oggetti.

CTreeCtrl e CTreeView

I due tipi di controlli si differenziano per il fatto che il primo e' indicato ad essere utilizzato in una dialog mentre il secondo in una vista in modo dinamico.
Questi tipi di controlli richiedono l' utilizzo di un'altra classe MFC che permette di gestire delle liste di immagini partendo da una sola composta da tante con le stesse dimensioni.
Quelle che seguono sono due esempi di bitmap multipli.
La classe CImageList permette di accedere tramite un indice alle singole immagini che compongono il bitmap.

Questa classe diventa utile quando si dispone di una grossa quantità di icone e immagini.



Le classi CTreeCtrl utilizzano le immagini per segnare i nodi dell' albero.
Come negli esempi precedenti create un progetto MFC basato su una dialog ed editatela aggiungendogli all' interno un Tree Control.
Richiamate il class wizard e selezionate il tab relativo alla gestione delle variabili.
Create una nuova variabile di tipo CtreeCtrl chiamata m_mytreectrl.
Successivamente andremo ad editare il file .H che contiene la dichiarazione della classe relativa alla dialog e ci aggiungeremo la dichiarazione delle variabili destinate a rappresentare i nodi dell' albero.

```
HTREEITEM     m_rghItem[12];
```

Nel nostro caso i nodi sono 12.
Inseriremo in una matrice le voci che inseriremo come descrizione verbosa dei nodi.
Normalmente tali descrizioni le potremmo ricavare dalla lettura, ad esempio, di records oppure da nomi di files.

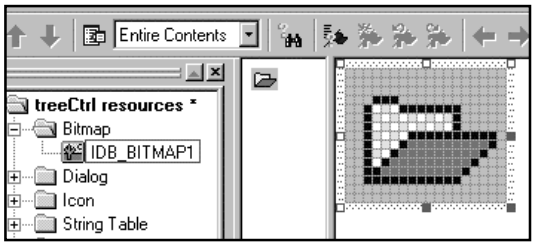
```
TCHAR  rgszItems[][18] =
        {_T("Ordine 1"), _T("Pezzo 1 Ordine 1"), _T("Pezzo 2 Ordine 1"), _T("Pezzo 3 Ordine 1"),
```

```
        _T("Ordine 2"), _T("Pezzo 1 Ordine 2"), _T("Pezzo 2 Ordine 2"), _T("Pezzo 3 Ordine 2"),
        _T("Ordine 3"), _T("Pezzo 1 Ordine 3"), _T("Pezzo 2 Ordine 3"), _T("Pezzo 3 Ordine 3")};
```

Dentro alla funzione InitDialog eseguiremo anche
Recuperate in qualche applicativo un bitmap come quelli mostrati in precedenza oppure create un semplice bitmap come ad esempio la seconda immagine della prima file.
Prima di terminare l'editing ricordatevi di guardare le dimensioni in quanto le dovrete fornire alle funzioni destinate alla gestione della lista.
Mediante il class wizard editate la funzione InitDialog e posizionatevi sul punto dove compare il TO DO.
Come prima cosa creeremo la CimageList che conterra' le immagini necessarie a rappresentare i nodi.
All' inizio della funzione eseguite le dichiarazioni che seguono:

```
CImageList *pimagelist;
Cbitmap bitmap1, bitmap2;
CCtrlIdemoApp  *pApp;
int iItem;
TV_INSERTSTRUCT tvstruct;
TCHAR rgszItems[][18] =
        {_T("Ordine 1"), _T("Pezzo 1 Ordine 1"), _T("Pezzo 2 Ordine 1"), _T("Pezzo 3 Ordine 1"),
        _T("Ordine 2"), _T("Pezzo 1 Ordine 2"), _T("Pezzo 2 Ordine 2"), _T("Pezzo 3 Ordine 2"),
        _T("Ordine 3"), _T("Pezzo 1 Ordine 3"), _T("Pezzo 2 Ordine 3"), _T("Pezzo 3 Ordine 3")};
```

Le seguenti due istruzioni creano un nuovo oggetto CimageList.
La prima linea costituisce il costruttore mediante l' operatore new.
La seconda istruzione invece indica che le immagini possiedono determinate coordinate.




```
pimagelist = new CImageList();
pimagelist >Create(17, 14, TRUE/*bMask*/, 1, 1);
```

Se utilizzate un immagine come quella mostrata qui sopra le dimensioni da usare nel metodo create sono 17 e 14.
Aggiungete le successive righe che servono ad aggiungere il bitmap per il nodo in radice relativo al numero dell' ordine.

```
Bitmap1.LoadBitmap(IDB_BITMAP1);
pimagelist->Add(&bitmap1, (COLORREF)0xFFFFFF);
```

Le due righe seguenti serviranno a settare il bitmap che servira' a rappresentare gli oggetti negli ordini che sono rappresentati da rami dell' albero derivanti da quelli in radice.

```
Bitmap2.LoadBitmap(IDB_BITMAP2);
pimagelist->Add(&bitmap2, (COLORREF)0xFFFFFF);
```

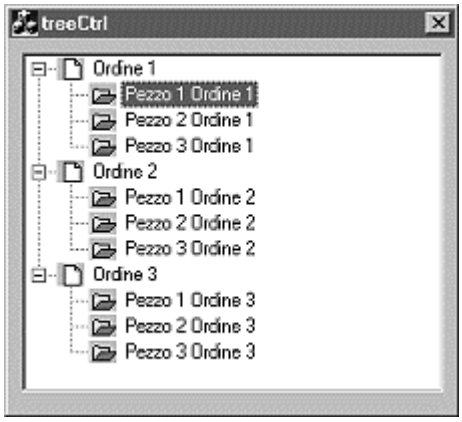
La prima riga lo legge dalle risorse mediante il suo ID mentre la seconda lo assegna alla CImageList.
Per essere precisi ci andrebbero due bitmap per ogni nodo in quanto il primo servirebbe a visualizzare il nodo in stato normale mentre il secondo lo dovrebbe visualizzare quando selezionato.
Nelle righe di codice esistono due assegnazioni destinate a risolvere questo caso.
Nel nostro esempio utilizzeremo sempre lo stesso bitmap per cui nell' assegnazione verranno utilizzati gli stessi valori.

```
for (iItem = 0; iItem < 12; iItem ++)
{
        tvstruct.hParent = (iItem % 4 == 0)? NULL : m_rghItem[iItem / 4 * 4];
        tvstruct.hInsertAfter = TVI_SORT;

        tvstruct.item.iImage = (iItem %4) ? 0 : 1;
        tvstruct.item.iSelectedImage =  (iItem %4) ? 0 : 1;

        tvstruct.item.pszText = rgszItems[iItem];
        tvstruct.item.mask = TVIF_IMAGE | TVIF_SELECTEDIMAGE | TVIF_TEXT;
        m_rghItem[iItem] = m_mytreectrl.InsertItem(&tvstruct);
}
```

Riporto a questo punto il codice di tutta la funzione InitDialog che contiene il codice generato dal class wizarde quello da noi inserito.

```
BOOL CTreeCtrlDlg::OnInitDialog()
{
	CImageList *pimagelist;
	Cbitmap bitmap1, bitmap2;
	int iItem;
	TV_INSERTSTRUCT tvstruct;
	TCHAR rgszItems[][18] = {_T("Ordine 1"), _T("Pezzo 1 Ordine 1"), _T("Pezzo 2 Ordine 1"), _T("Pezzo 3 Ordine 1"),
		_T("Ordine 2"), _T("Pezzo 1 Ordine 2"), _T("Pezzo 2 Ordine 2"), _T("Pezzo 3 Ordine 2"),
		_T("Ordine 3"), _T("Pezzo 1 Ordine 3"), _T("Pezzo 2 Ordine 3"), _T("Pezzo 3 Ordine 3")};

	// NON E' RIPORTATO IL CODICE CREATO DAL WIZARD
	// ……..
	// TODO: Add extra initialization here

	pimagelist = new CImageList();
	pimagelist->Create(17, 14, TRUE/*bMask*/, 1, 1);

	bitmap1.LoadBitmap(IDB_BITMAP1);
	pimagelist->Add(&bitmap1, (COLORREF)0xFFFFFF);

	bitmap2.LoadBitmap(IDB_BITMAP2);
	pimagelist->Add(&bitmap2, (COLORREF)0xFFFFFF);

	m_mytreectrl.SetImageList(pimagelist, TVSIL_NORMAL);

	for (iItem = 0; iItem < 12; iItem ++)
	{
		tvstruct.hParent = (iItem % 4 == 0)? NULL : m_rghItem[iItem / 4 * 4];
		tvstruct.hInsertAfter = TVI_SORT;
		tvstruct.item.iImage = (iItem %4) ? 0 : 1;
		tvstruct.item.iSelectedImage =  (iItem %4) ? 0 : 1;
		tvstruct.item.pszText = rgszItems[iItem];
		tvstruct.item.mask = TVIF_IMAGE | TVIF_SELECTEDIMAGE | TVIF_TEXT;
		m_rghItem[iItem] = m_mytreectrl.InsertItem(&tvstruct);
	}

	return TRUE;  // return TRUE  unless you set the focus to a control
}
```

Il membro hParent della struttura utilizzata per inserire gli oggetti contiene l' handle dell' oggetto genitore oppure NULL o TVI_ROOT se l' oggetto deve essere inserito nella radice.
hInsertAfter invece contiene l' handle dell' oggetto dopo il quale quello nuovo deve essere inserito.
Anche in questo caso e' possibile inserire dei valori costanti e precisamente :
TVI_FIRST se deve essere inserito all' inizio della lista, TVI_LAST se deve essere inserito in fondo o TVI_SORT se l' oggetto deve assumere una posizione in base all' ordine dello stesso nella lista.
Fate attenzione che ho eliminato il codice creato nell' initdialog dal wizard.
In pratica si tratta dalla dichiarazione delle variabili al TO DO :



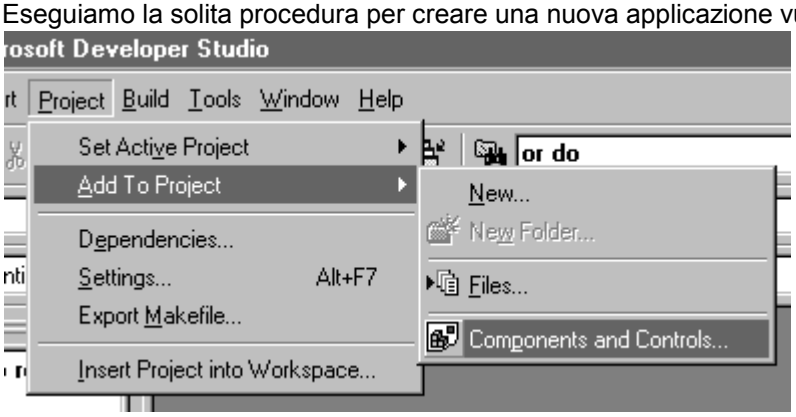
Compilando il tutto avremo a questo punto il nostro CTreeCtrl funzionante dentro la dialog come nella finestra che segue.
Nella classe CTreeCtrl esistono moltissimi altri metodi per le funzioni piu' svariate che servono all' organizzazione del controllo del tipo DeleteItem, GetItem ecc.
Tramite class wizard possibile mappare i messaggi per gestire tutti gli eventi che possono essere generati lavorando su questi controlli.
Sono inoltre disponibili inoltre delle costanti che permettono di stabilire lo stile del controllo come adesempio se deve disporre delle linee che uniscono i nodi, se deve comparire il segno che indica se il nodo e' espanso o meno ecc.

Se utilizzate questo tipo di controllo da resource editor potete settare questi valori direttamente senza dover utilizzare dopo funzioni particolari.

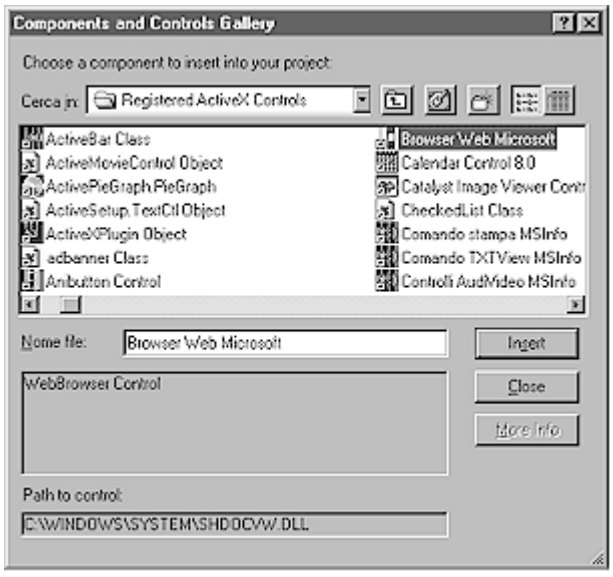
I parametri settabili da resource editor sono quelli mostrati nella successiva immagine.

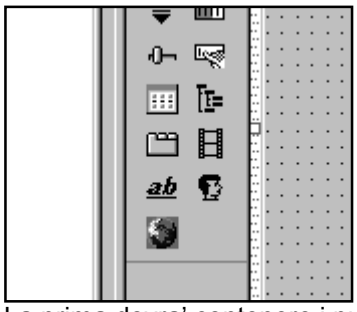
IWebBrowser e IWebBrowserApp

WebBrowser e' un controllo Active X che gli sviluppatori possono utilizzare per inserire funzioni di browsing all' interno delle proprie applicazioni.
E' possibile utilizzarlo da diversi linguaggi Microsoft anche se noi ne vedremo l' uso mediante Visual C++.
IWebBrowserApp in pratica e' un super set di IWebBrowser.
Come nel caso degli altri esempi scriveremo un piccolo applicativo che mostri l' utilizzo di Class Wizard per la scrittura di programmi C++.
Vedremo inoltre come aggiungere funzioni di stampa al Browser.
Alcune tecniche viste fino a questo punto potrebbero essere abbinate per creare un programma piu interessante sia dal punto di vista estetico che funzionale.
Per attenerci a quel clima di semplicita' dichiarato inizialmente vedremo soltanto la tecnica di base senza inserire complicazioni progettuali che potrebbero comunque essere inserite una volta imparato il metodo di utilizzo.
Eseguiamo la solita procedura per creare una nuova applicazione vuota utilizzando il class wizard.



Creiamo l' applicazione dialog based. Avremmo potuto anche creare un applicazione SDI o MDI e poi inserirgli un dialog costruita con dentro il controllo destinato alla gestione del Browser e i vari pulsanti e controlli legati alla navigazione.
Come primo passo dovremo includere nel nostro applicativo il controllo Active X.
Selezioniamo Project da menu e successivamente Add to Project e Componets and Controls.
Fatto questo comparira' la maschera da cui sara' possibile selezionare se inserire un componente di Developper Studio o un controllo Active X registrato.
Selezioniamo Registered ActiveX Controls e andiamo a cercare il controllo Browser Web Microsoft.

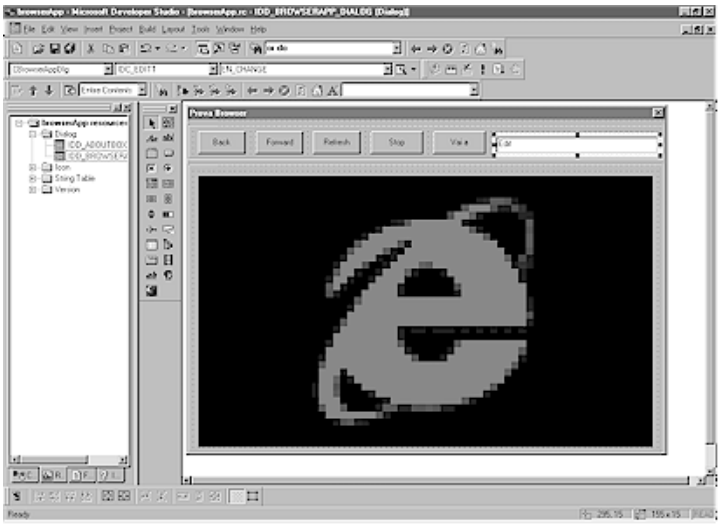




Nel vostro progetto verranno aggiunti i file webbrowser2.cpp e webbrowser2.h.
Bisognera' inoltre ricordarsi, nel caso che si tratti di un applicazione da distribuire, che e' necessaria la presenza della DLL SHDOCVW.DLL che in pratica e' quella che contiene tutti i metodi utilizzati dal browser.
A questo punto editiamo la dialog che era stata creata dal wizard.

Nella cartella degli oggetti ne troveremo uno in piu' relativo all' oggetto browser.
Ora dovremo soddividere la dialog in due parti.
La prima dovra' contenere i pulsanti e il campo in cui specificare l' indirizzo da raggiungere.
La seconda sara' l' area in cui verranno visualizzate le pagine.
Quella che segue potrebbe essere una soluzione di layout della nostra dialog.
Quella che segue e' la tabella che riporta l' elenco dei metodi di IWebBrowser preso dalla documentazione Microsoft.

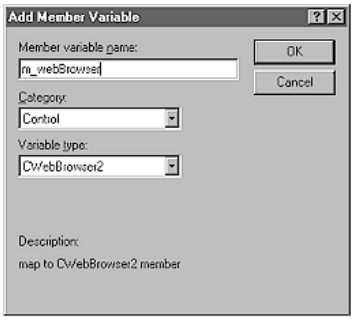
IWebBrowser Methods	**Description**
GoBack	Navigates to the previous item in the history list.
GoForward	Navigates to the next item in the history list.
GoHome	Navigates to the current home or start page.
GoSearch	Navigates to the current search page.
Navigate	Navigates to a resource identified by a Universal Resource Locator (URL)
Refresh	Reloads the current file.
Refresh2	Reloads the current file and optionally prevents the pragma:noCache header from being sent.
Stop	Stops opening a file.
get_Application	Returns an Application object representing the application that contains the Web browser control.
get_Parent	Returns the form on which the Web browser control is located.
get_Container	Returns the container of the Web browser control, if any.
get_Document	Returns the active document.
get_TopLevelContainer	Returns a value indicating whether the current object is the top level container of the Web browser control.
get_Type	Returns the type of the current contained object
get_Left	Returns the distance between the internal left edge of the Web browser control and the left edge of its container.
put_Left	Sets the distance between the internal left edge of the Web browser control and the left edge of its container.
get_Top	Returns the distance between the internal top edge of the Web browser control and the top edge of its container.
put_Top	Sets the distance between the internal top edge of the Web browser control and the top edge of its container.
get_Width	Returns the horizontal dimension of the frame window that contains the Web browser control.
put_Width	Sets the horizontal dimension of the frame window that contains the Web browser control.
get_Height	Returns the vertical dimension of the frame window that contains the Web browser control.
put_Height	Sets the vertical dimension of the frame window that contains the Web browser control.
get_LocationName	Returns the name of the resource that the Web browser control or Internet Explorer is currently displaying.
get_LocationURL	Returns the URL of the resource that the Web browser control or Internet Explorer is currently displaying.
get_Busy	Returns a value indicating whether a download or other activity is still in progress.



Dopo aver creato la dialog dovremo utilizzare i metodi appena elencati per creare le funzioni necessarie alla navigazione.
In alcuni casi la cosa e' semplicissima.
Prendiamo ad esempio il pulsante Back.
Sara' sufficiente creare la funzione, mediante il class wizard, e dopo averla edita inserirgli :

```
var_web_browser.GoBack();
```

Creiamo la variabile relativa al WebBrowser.
Attiviamo il class wizard e selezioniamo il tabulatore perdefinire una nuova variabile.
Selezioniamo l' ID del WebBrowser che se non l' avete modificato rispetto a quello proposto dovrebbe essere



IDC_EXPLORER1.
Premiamo il tasto Add Variable e assegniamo il nome m_webBrowser al controllo.
A questo punto creiamo le funzioni che verranno abbinate ai vari tasti.
Attiviamo il class wizard e selezioniamo il Messages Map.
Mettiamo in risalto l' id relativo al pulsante Back, selezioniamo il messaggio BN_CLICKED e premiamo il tasto Add Function.
Nominiamo la funzione OnBack().
Dopo averla creata editiamola con premendo Edit Function ed inseriamogli il seguente codice.

```
void CBrowserAppDlg::OnBack()
{
	m_webBrowser.GoBack();
}
```

Lo stesso dicasi per il tasto Forward.
In questo caso il codice sara' :

```
void CBrowserAppDlg::OnForward()
{
	m_webBrowser.GoForward();
}
```

Per il tasto Stop e per quello Refresh la questione e' simile in quanto e' sufficiente inserire i metodi che svolgono le relative funzioni e precisamente Stop() eRefresh().
Se avessimo inserito il tasto Go Home avremmo dovuto inserire anche in questo caso un solo metodo e precisamente GoHome().
Alcuni parametri vengono stabiliti tramite le opzioni di Internet Explorer.
Prendiamo per esempio la funzione GoSearch() che va alla pagina relativa la motore di ricerca (anche per la funzione appena vista GoHome() e' la stessa cosa).
I valori vengono stabiliti mediante la specifica delle proprieta' di Internet Explorer.
Per la funzione Vai a dovremo invece leggere l' indirizzo dall' apposito campo di edit e utilizzarlo nella funzione Navigate().
Questo metodo possiede la seguente sintassi.

```
HRESULT IWebBrowser::Navigate(BSTR URL, // URL to navigate to
VARIANT FAR* Flags, // address of option flags
VARIANT FAR* TargetFrameName, // address of frame name for resource
VARIANT FAR* PostData, // address of HTTP POST data
VARIANT FAR* Headers, // address of HTTP headers);
```

Il primo parametro e' la stringa che rappresenta l' indirizzo.

Il secondo invece e' l' indirizzo di una variabile nella quale sono specificati i parametri che possono essere una combinazione di quelli definiti come BrowserNavConstant e precisamente:

```
typedef enum BrowserNavConstants {
   navOpenInNewWindow = 1,
   navNoHistory = 2,
   navNoReadFromCache = 4,
   navNoWriteToCache = 8
} BrowserNavConstants;
```

Elements

navOpenInNewWindow	Open the resource or file in a new window.
NavNoHistory	Do not add the resource or file to the history list. The new page replaces the current page in the list.
NavNoReadFromCache	Do not read from the disk cache for this navigation.
NavNoWriteToCache	Do not write the results of this navigation to the disk cache.

Il terzo parametro e' relativo ad una stringa che contiene il nome del frame nel quale visualizzare la risorsa.
Il quarto serve per i trasferimenti di dati verso un form html.
L' ultimo infine rappresenta il valore di un header http che deve essere inviato verso un server.
A questo punto non ci rimane che editare tramite class wizard la funzione Initdialog per inserirgli il metodo Navigate() con un indirizzo iniziale di prova.
Nel mio caso ho inserito l' indirizzo di un WEB locale creato con Xitami WEB Server (simile al Personal Web Server di Microsoft) che utilizzo per fare prove in locale.
La riga aggiunta al TO DO: della funzione InitDialog e' la seguente (voi inserite l' indirizzo che preferite).

```
m_webBrowser.Navigate("http://flavio", NULL, NULL, NULL, NULL);
```

Provate ad eseguire la prima compilazione e provate a vedere il risultato di quanto fatto fino ad ora.
Eseguiamo la creazione della variabile che ci servira' a reperire il valore specificato nel campo di edit.
Attiviamo il class wizard e creaiamo una nuova variabile legata all' ID del campo di edit.
Chiamiamolo m_Indirizzo di tipo Cstring.
Ora creiamo la funzione legata al tasto Vai A.
Sempre dal class wizard attiviamo il tabulatore su Message Maps.
Selezioniamo l' identificatore del pulsante Vai a e premiamo il tasto per creare una nuova funzione.
Chiamiamo la funzione OnVaia ed editiamola per inserirgli il seguente codice:

```
void CBrowserAppDlg::OnVaia()
{
	UpdateData(TRUE);
	m_webBrowser.Navigate(m_Indirizzo, NULL, NULL, NULL, NULL);
}
```

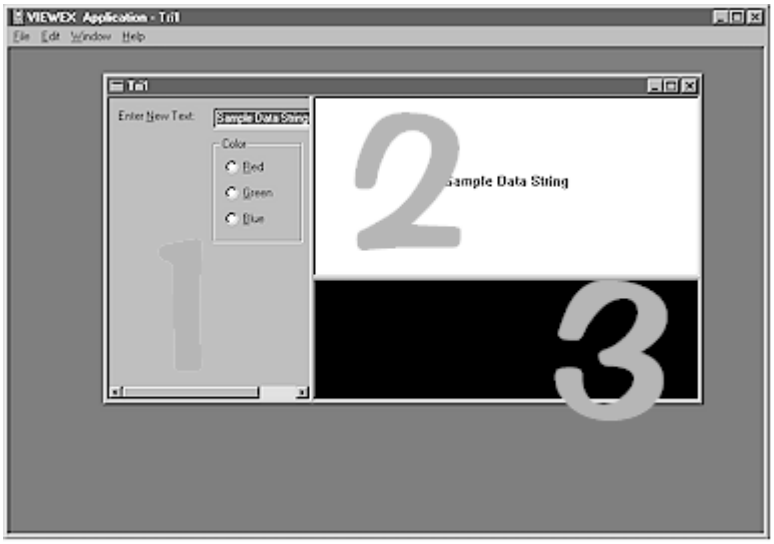
Ci tengo a ricordare che in questaserie di articoli l' esenzialismo e' volontario per eviatre complicazioni che potrebbero essere d' intralcio per la comprensione dei concetti fondamentali.
Una volta appresi questi ultimi tutte le aggiunte potranno essere inserite nei vostri progetti reali.

SPLITTER WINDOWS

Esistono dei casi in cui risulta utile poter suddividere la finestra in piu' parti ciascuna delle quali contenente gli oggetti necessari per determinate funzionalità.
Per portare un esempio pratico mi riferirò ad un problema che ultimamente ho risolto mediante l' utilizzo delle funzioni atte alla gestione delle finestre splittate.

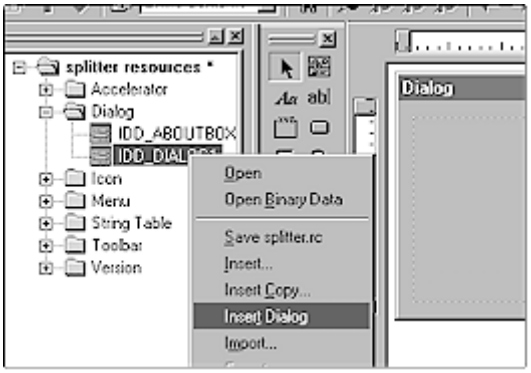
Per gestire dei programmi che permettessero l' accesso ad internet a persone con un account a disposizione avevo la necessita' di avere il video suddiviso in due parti.

In una porzione doveva esserci la maschera che permetteva di fare il login mentre nell' altra parte ci doveva essere il browser per la navigazione.
Nei tempi morti, ovvero quelli in cui nessuno utilizzava la macchina, nella porzione destinata al browser dovevano ciclare alcune pagine locali con pubblicità di ditte.
In pratica il video doveva essere suddiviso nel seguente modo.



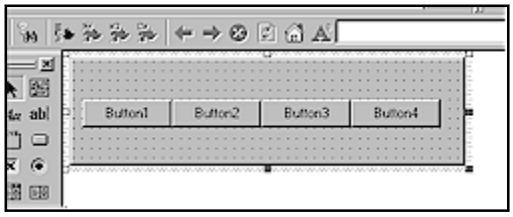
Come potete notare le due zone video, in questo caso, dispongono di scrollbar destinate a gestire lo scorrimento relativo ad ogni zona in modo indipendente.
Fate attenzione che non si tratta di un applicazione MDI in cui nella client area vengono aperte più finestre indipendenti.
Qui si tratta di una sola window suddivisa in piu' parti (nell' esempio due).
La seguente immagine e' relativa ad un esempio
Microsoft in cui la suddivisione e' in tre parti.
Inoltre la cosa interessante e' che e' possibile associare alle zone suddivise delle dialog appositamente create a parte.
Le zone in cui viene suddivisa la window derivano da Cview ma possono derivare anche da qualsiasi oggetto CWnd che possiedono un ID appropriato come child window.
Una CSplitterWnd in genere e' contenuta in oggetti CFrameWnd oppure CMDIChildWnd.
La creazione avviene in genere in tre passi e precisamente i seguenti.

1. Creazione della variabile CsplitterWnd dentro alla classe della parent frame.
2. Override della funzione CframeWnd::OnCreateClient.
3. Inserimento dei metodi necessari alla creazione della CsplitterWnd e delle viste da inserire in questa.

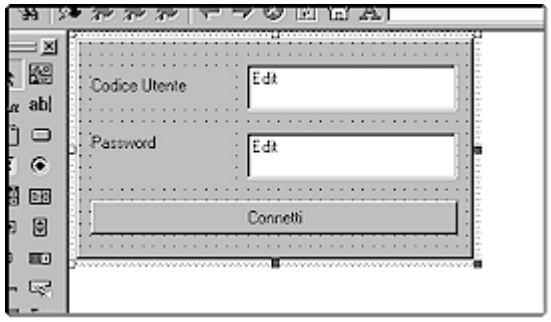


Mediante due apposite funzioni e' possibile creare una CsplitterWnd dinamica o statica.
Vediamo comunque passo a passo l' utilizzo di tali funzioni.
Supponiamo di dover disporre di due porzioni di finestra nelle quali vengono inseriti i comandi necessari ad eseguire un login, nella prima, e quelli necessari a processare dei dati nella seconda.
Attiviamo Visual C++ e creiamo mediante l' application Wizard un progetto chiamato SPLITTER.
A questo punto creiamo le due dialog che utilizzeremo come viste nelle due porzioni di finestra utilizzando il resource editor.
Creiamo la prima dialog utilizzando l' opzione Insert Dialog nella gestione risorse.
A livello di esempio inseriremo solo due LABEL , due CAMPI di inserimento e un pulsante anche se potete comporla come piu' vi piace.
Mediante l' opzione Properties settiamo la dialog senza caption e senza system menu.

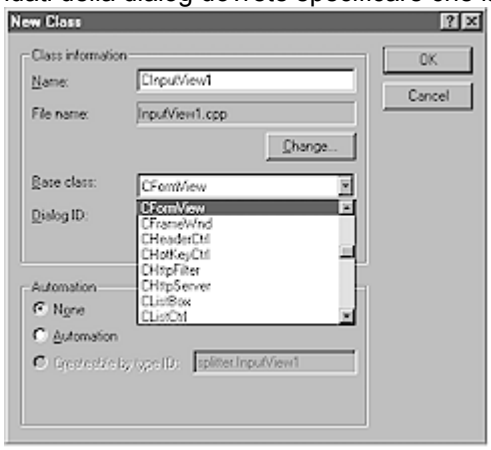
Inoltre selezioniamo lo stile CHILD.



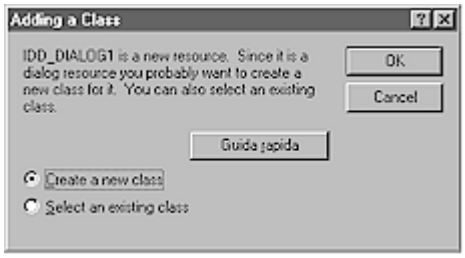
La dialog avra' il seguente layout.



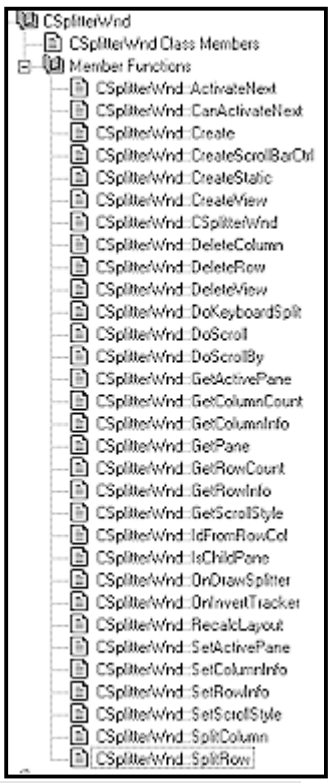
Durante questa fase relativa alla creazione della dialog relativa aduna porzione della finestra utilizzando il Class Wizard creeremo anche la classe relativa alla dialog stessa.
Appena selezionata la voce del menu relativo alla ClassWizard Visual C++ si accorgera' che non esiste nessuna classe relativa alla dialog e quindi vi richiedera' se volete crearla.
Accettate quanto vi propone e chiamate la nuova classe CInputView1.
Quando vi verra' mostrata la maschera in cui sono richiesti idati della dialog dovrete specificare che la vostra nuova classe deriva da CformView.

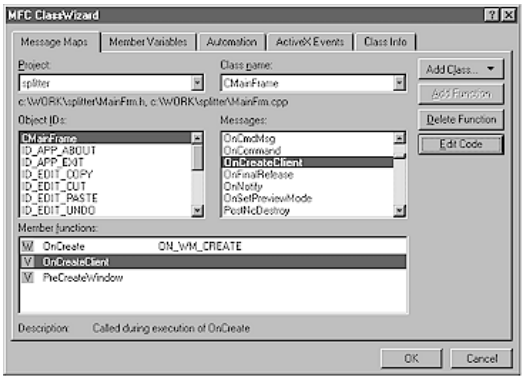


A uesto punto seguendo la stessa metodologia creerete anche l' altra dialog .



In questo caso la classe abbinata si chiamera' CinputView2 e la sua classe di base sara' sempre CformView.
Potete comporre la dialog come volete includendo in questa anche controllo ActiveX.
Per non rendere lungo l' esempio inseriro' nella dialog soltanto quattro pulsanti.
Le classi create potrebbero contenere tutti i metodi relativi alla gestione della dialog.
In queste esempio non vengono neppure abbinati metodi ai quattro pulsanti.
A questo punto le funzioni necessarie a creare la suddivisione della window e le viste devono essere utilizzate eseguendo l' override del metodo OnCreateClient.
Aprite in edit il file che gestisce il frame che intendete splittare.
Se avete seguito l'esempio il file e' MainFrm.cpp.





Attivate il class wizard e selezionate il tab Message Maps.
Nella casella di selezione Class name selezionate CmainFrame e create una nuova funzione per il messaggio OnCreateClient.
A questo punto editeremo il messaggio e utilizzando le apposite funzioni creeremo la suddivisione e le viste.
La lista dei metodi della classe MFC che gestisce questa funzionalita' sono le seguenti.
Prima di modificare OnCreateClient inserite nella classe CminFrame, definita in MainFrm.h, splitterWnd.

```
CSplitterWnd m_wndSplitter;
```

La funzione diventa :

```
BOOL CMainFrame::OnCreateClient(LPCREATESTRUCT lpcs, CCreateContext* pContext)
```

```
{
	if (!m_wndSplitter.CreateStatic(this, 1, 2))
	{
		TRACE0("Failed to CreateStaticSplitter\n");
		return FALSE;
	}
	if (!m_wndSplitter.CreateView(0, 0,RUNTIME_CLASS(CInputView1), CSize(250, 50), pContext))
	{
		TRACE0("Failed to create first pane\n");
		return FALSE;
	}
	if (!m_wndSplitter.CreateView(0, 1, RUNTIME_CLASS(CInputView2), CSize(250, 50),
pContext))
	{
		TRACE0("Failed to create second pane\n");
		return FALSE;
	}
	SetActiveView((CView*)m_wndSplitter.GetPane(0,1));
	return TRUE;
}
```

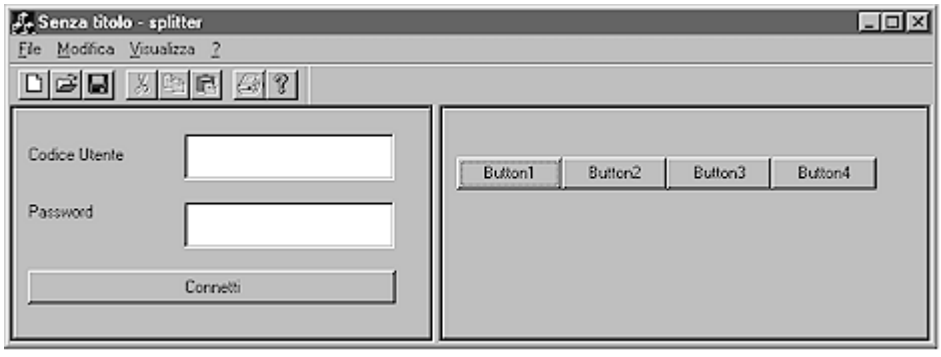
La creazione delle due classi CinputView1 e CinputView2 relative alle due dialog create ora utilizzate per la creazione delle views ha generato i file InputView1 e InputView2 con i relativi files .h.
Per poter utilizzare le due classi nelle funzioni per l' abbinamento dovete includere nel file MainFrm.cpp i due files

```
#include "InputView1.h"
#include "InputView2.h"
```

Prima di compilare il tutto editate con il class wizard la funzione PreCreateWindow nel modulo MainFrm.cpp e utilizzate la CREATESTRUCT cs passata come argomento per settare le dimensioni della finestra principale.
La funzione diventera' :

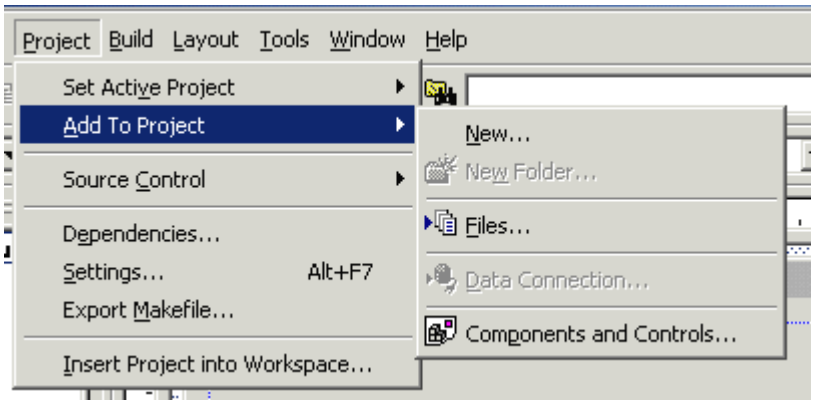
```
BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	cs.cx = 650;
	cs.cy = 250;

	return CFrameWnd::PreCreateWindow(cs);
}
```

Ora compilate ed eseguite.
Il risultato sara' :

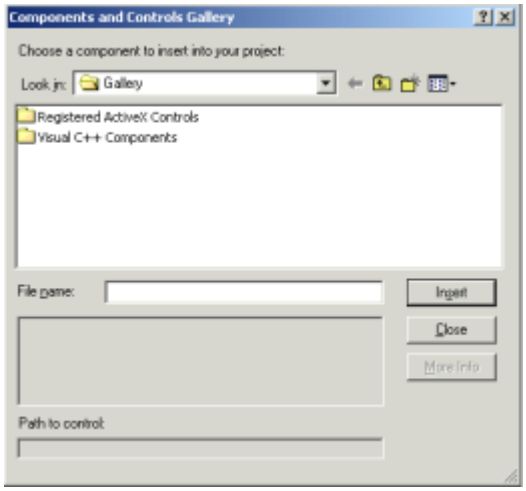


Altre tecniche

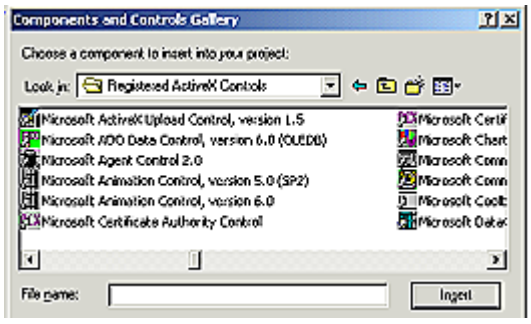
L'introduzione di Windows e le metodologie di programmazione Object Oriented (della serie è nato prima l'uovo o la gallina) hanno costituito la base per la creazione di alcuni sistemi comuni che possono essere utilizzati da tutti i linguaggi di programmazione.
All'inizio la tecnologia COM (Component Object Model) è stata utilizzata come base per la creazione di nuove tecnologie ActiveX.
Al giorno d'oggi tali tecnologie sono comunemente implementate in Microsoft Windows oltre ad essere Anche parte integrante dei linguaggi Visual come ad esempio Visual C++.
Visual C++ dispone della possibilità di implementare all'interno dei propri programmi oggetti derivati da sistemi ActiveX.
Questi una volta importati possono servire per la dichiarazione di oggetti con a sua disposizione proprietà e metodi utilizzabili per controllare le funzionalità nell'ambito dei programmi.
Finiremo questo breve testo portando come esempio la gestione di un database tramite funzionalità ADO incluse dentro al programma tramite importazione di oggetti ActiveX.
Nelle pagine prima avevamo visto l'utilizzo delle classi DAO per la gestione di database.
Ora mediante l'activeX di ADO potremo gestire database senza molto codice.
Come al solito creiamo un nuovo progetto MFC , orientato alla DIALOG, chiamandolo gestData.
A questo punto inseriamo dentro alla toolbar degli oggetti per la dialog gli ActiveX destinati ala gestione del database.
Ora selezioniamo da menu PROJECT->ADD TO PROJECT->COMPONENTS AND CONTROLS



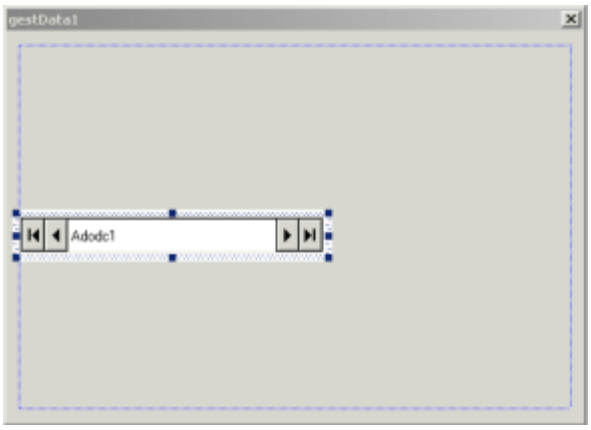
Ci verrà mostrata la finestra da cui potremo selezionare I vari ActiveX installati sul nostro sistema che potremo includere dentro al nostro programma.



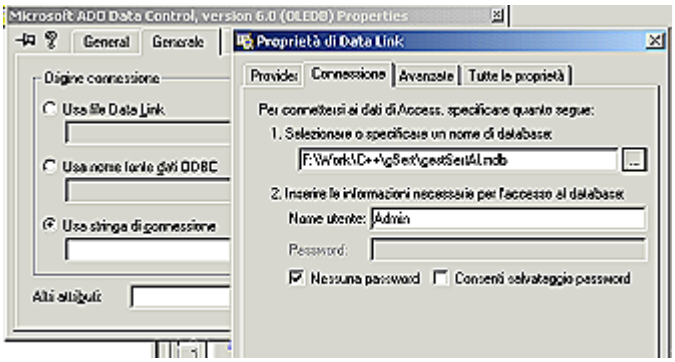
Selezioniamo Registered ActiveX Controlls e poi Microsoft ADO Control Version 6.0 (OLEDB), Microsoft Datagrid Controls Version 6.0 (OLEDB) e Microsoft Masked Edit Control, Version 6.0.
Per ciascuno di questi inseriti il Visual C++ ci chiederà se voliamo inserire un certo numero di classi.
Chiaramente rispondiamo di si.
La nostra TOOLBAR verrà mostrata con TRE oggetti nuovi.
Questi verranno utilizzati per inserire i controlli ati a gestire il database sulla nostra dialog.



Ora inseriamo il primo oggetto sulla dialog dopo aver eliminato tutti quelli che il generatore aveva messo di default.



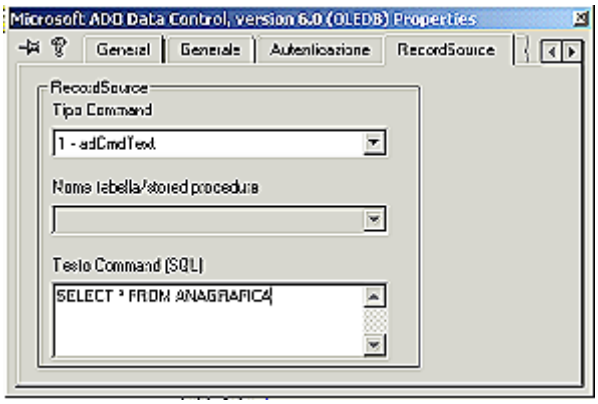
Il primo è la barra di navigazione sui record quella che di fatto esegue il collegamento con il database vero e proprio.
Per eseguire questo collegamento premendo il tasto destro del mouse richiediamo di editare le proprietà.
Nel tabulatore della dialog che ci compare selezioniamo Generale dopo il metodo di collegamento al database.
Clickiamo su USA STRINGA DI CONNESSIONE e poi dalla lista di scelta che ci compare Microsoft Jet 4.0 OLE DB.
A questo punto dovremo selezionare il nome del file DB da usare che dovrà essere in formato ACCESS, ovvero un file .MDB.



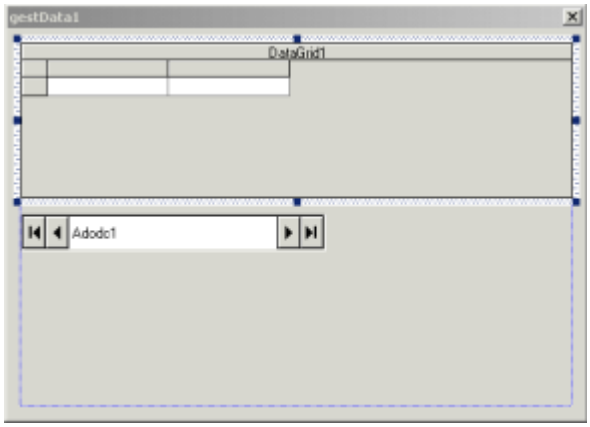
Nelle proprietà avanzate selezioneremo anche READ/WRITE e poi premeremo OK.
.
A questo punto dovremo fare forse l'operazione più complessa.
Questa SCROLLBAR collegata al database verrà utilizzata come sorgente dei dati agli altri oggetti orientati alla visualizzazione di questi.
Come saprete un database ACCESS potrebbe contenere più files per cui in questo ambito dovremo anche dirgli come selezionare i dati.
Nel tabulatore RECORD SOURCE potremo specificare uno statement SQL che verrà usato da filtro.
In pratica sapendo che all'interno del data base c'e' il file ANAGRAFICA, ad esempio, potremmo dare :

SELECT * FROM ANAGRAFICA

In questo caso tutti gli oggetti che useranno questa TOOLBAR come origine dati utilizzeranno i dati filtrati da questi statement.

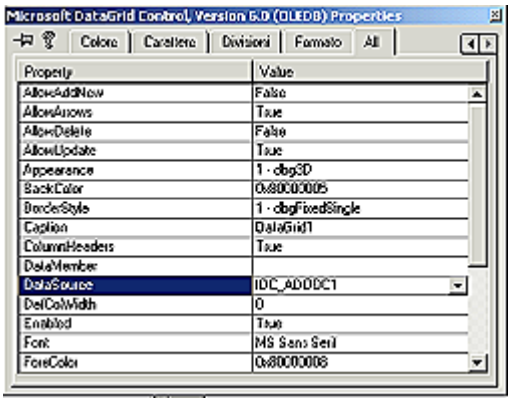


Ora posizioniamo la lista da cui sarà possibile selezionare i record
Scegliamo l'oggetto dalla TOOLBAR.
Posizioniamo a video la tabella in modo che esteticamente sia accettabile.
Ora con il tasto destro del mouse selezioniamo le proprietà della lista appena messa sulla dialog.

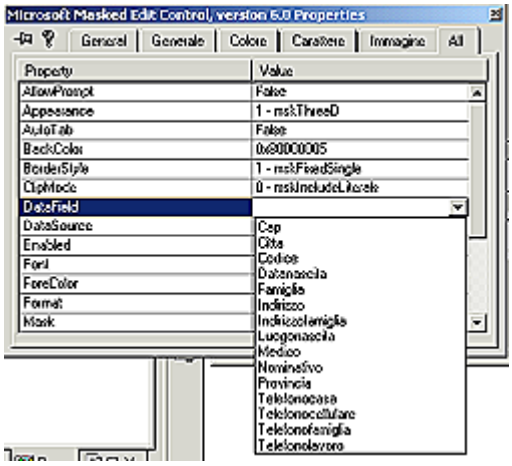


Nell'ultima tabulazione della dialog che ci compare ci sono tutte le proprietà da cui potremo selezionare la fonte dati a cui desideriamo collegarci.
Questa è quella rappresentata dalla barra di scorrimento posizionata prima.
Il nome di default data da visual C++ è IDC_ADOC1.
A questo punto la lista sarà relazionata alla barra di scorrimento che noi prima avevamo associato al

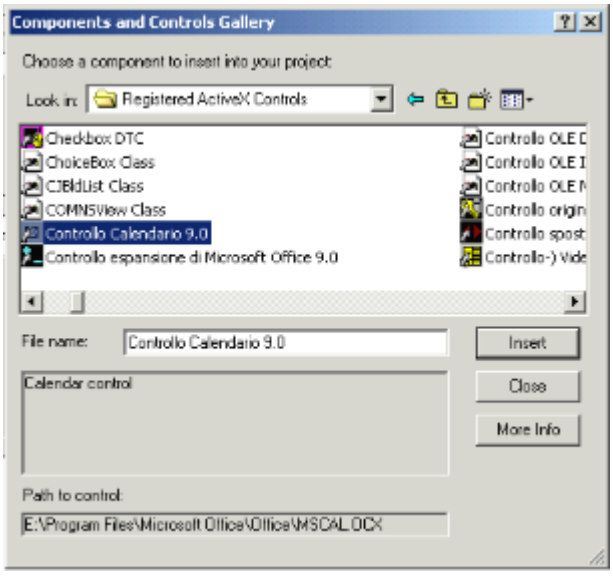
database.
Ricordatevi che i record visualizzati saranno queli filtrati dallo statement SQL che avevamo settato nelle proprietà della barra di scorrimento records.



Ora potremmo richiedere a Visual C++ di farci vdere un preview della dialog che abbiamo creato fino ad adesso.
Vedremmo già i dati contenuti nella tabella.
Quando avevamo selezionato i tre ActiveX ne avevamo messo un terzo ovvero quello relativo al MASK EDIT.
Questo permette di editare ogni singolo campo al contrario degli oggetti posizionati fino ad ora i quali servivano, il primo a gestire il collegamento con il database e lo scorrimento dei records, mentre il secondo a visualizzare ij formato gabellare i dati dell'archivio scelto.
Ora se volessimo editare singolarmente i vari valori dei campi potremmo mettere tanti campi di EDIT MASK quanti sono quelli che vogliamo inserire sulla dialog.

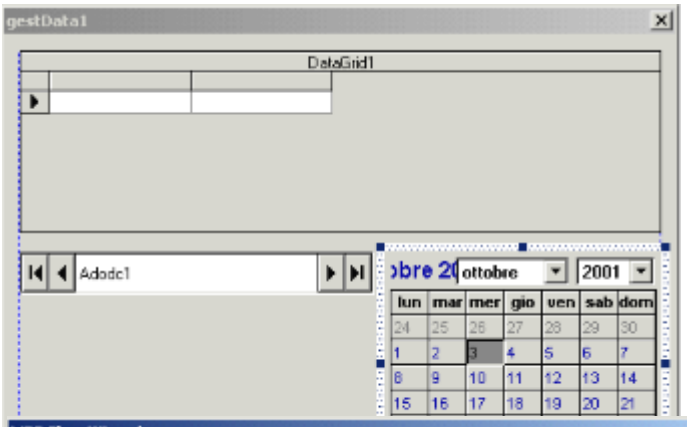


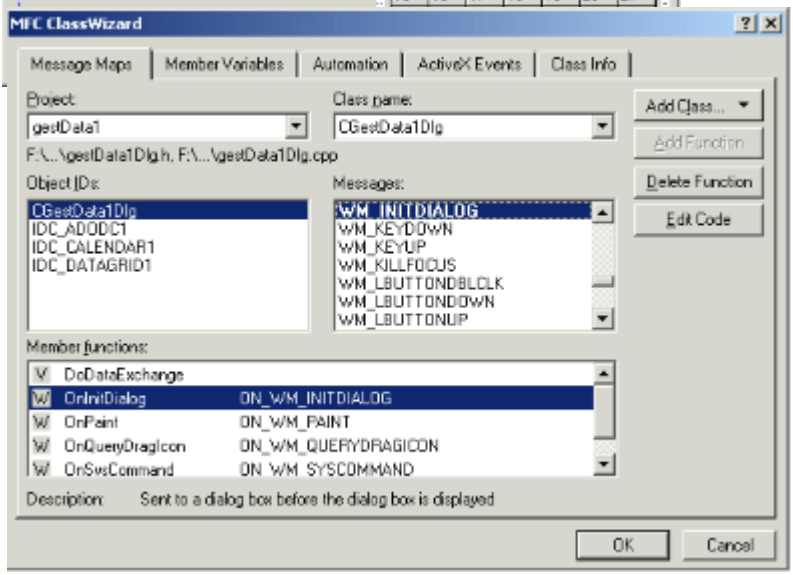
Per fare l'esempi ne posizionerò soltanto uno.
Nelle proprietà dovremo selezionare la fonte del database (sempre IDC_ADOC1) ma anche quale campo vorremo associare al campo.
Sempre richiedendo il PREVIEW della dialog vedremo il suo comportamento.
Ora se vorreste curare maggiormente il database potreste metterci i campi statici di descrizione dei vari valori associati ai campi d'edit, ed altre funzioni.
Quello che mi interessava farvi vedere era semplicemente come era possibile sfruttare anche ActiveX per aggiungere funzionalità ai vostri programmi.
In altre parole Visual C++ possiede diverse classi nelle librerie MFC e grazie alla dichiarazione di variabili legate queste potete implementare funzionalità che vi servono a svolgere certi compiti.
Nel caso in cui le librerie di Visual C++ non possiedano le funzionalità che vi servono potrete sempre cercare degli ActiveCX che svolgono certi compiti e utilizzarli come abbiamo usato gli oggetti ADO nell'esempio appena visto.
Per farvi un esempio in uno degli ultimi programmi che mi sono stati commissionati si trattava di implementare funzioni di grafica molto complesse.
In pratica si trattava di riscrivere una specie di CorelDraw.
Scrivere interamente un programma come Corel Draw è una cosa molto complessa.
Nel giro di un mese l'ho scritto in quanto ho comprato un modulo ActiveX che aveva la possibilità di gestire un foglio di disegno, posizionarci oggetti grafici come linee, rettangoli, cerchi ecc., settargli gli attributi, i riempimenti, le ombreggiature, gli effetti ecc.
Vogliamo riportare un altro esempio ?
Bene.
Supponiamo che per qualsiasi motivo sulla dialog di prima dobbiamo fare in modo che l'utente selezioni una data.



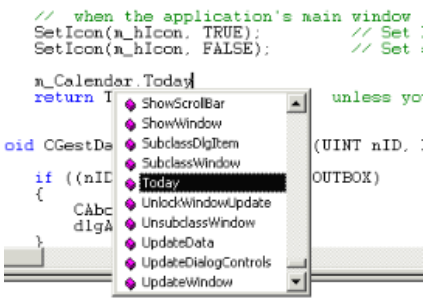
Potremmo inserire un semplicissimo campo d'edit controllando poi, dopo l'input da parte dell'utente, il formato corretto della data oppure potremmo utilizzare un AqctiveX legato alla gestione de calendari.
Come avevamo scelto gli ActiveX per la gestione del database seguiamo la stessa procedura ma questa volta selezioniamo il controllo Controllo Calendario 9.0.
Allo stesso modo di prima una nuova icona si aggiungerà alla TOOLBAR degli oggetti che possiamo inserire nella nostra dialog.
Ora inseriamo questo nuovo oggetto sulla dialog.
Questa volta dovremo dichiarare tramite il Class Wizard, una variabile che identificherà questo

calendario in modo tale che potremo richiamare i suoi metodi al fine di controllarne le funzionalità.
Premiamo il tasto destro del mouse e selezioniamo il class wizard.
Selezioniamo il tabulatore Member Variables e dopo aver selezionato l'oggetto calendario premiamo il pulsante legato alla creazione di una nuova variabile.
Chiamiamo la variabile m_Calendar.
Ora quando la dialog viene creata viene creato l'evento WM_INITDIALOG ovvero Windows in fase di creazione della dialog lancerà un messaggio di questo tipo.
Se inseriamo una funzione adatta a gestirla potremmo inserire dentro a questa delle funzioni di inizializzazione degli oggetti.
Nel caso del calendario potremmo usare il metodo incluso nella classe Today() il quale setta la data attuale nel calendario.
Per gestire il messaggio di INITDIALOG selezioniamo sempre il Class Wizard ma scegliendo il tabulatore Message Maps.





Selezioniamo l'identificatore della dialog e poi, nell'apposita lista, andiamo a cercare il messaggio di WM_INITDIALOG.
Dopo aver creato la funzione, se già non esiste, editiamola e aggiungiamo il codice che segue :



m_Calendar.Today()

Visual C++ possiede un sistema sensitivo che mano mano che scriviamo ci proporrà tutte le possibili scelte.
Ora quando sarà necessario leggere la data selezionata potremo sempre usare dei metodi relativi alla classe calendario stessa come ad esempio GetDay(), GetMonth(), GetYear().

Con questo concludo il tutto sperando che il testo sia servito allo scopo.
Per qualsiasi altro libro da me scritto riferitevi sempre al sito :

www.crackinguniversity2000.it

Note sul copyright

Ho sempre nel limite del possibile rispettato la filosofia legata al public domain ma a patto che tale rispetto avvenga anche per quanto riguarda le volontà degli autori nell'ambito di quanto distribuito.
Tale testo può essere distribuito in modo FREE a patto che non vengano modificati i dati dell'autore.
L'uso di questo non è vincolato a nessun pagamento a patto che il suo fine non sia quello del lucro.
Potete utilizzarlo anche in ambiti di corsi ma solo dopo averne fatto esplicita richiesta il versamento tramite vaglia, invio diretto ecc. di un offerta libera darà diritto a ricevere automaticamente via email tutti in nuovi testi dell'autore.
Scritti da Flavio Bernardotti sono disponibili :

Linguaggio C a basso livello	400 pagine	1986
Linguaggio C tricks & tips	100 pagine	1987
Programmazione in Linguaggio C in ambiente Unix	150 pagine	1988
Linguaggio SQL	100 pagine	1988
Programmazione in Java	270 pagine	1999
Cracking & Hacking	1000 pagine	2000

Protocolli di rete	60 pagine	2001
Introduzione al linguaggio C/C++	60 pagine	2001

Flavio Bernardotti
Project Manager at WEBSITEK.COM
Parco Scientifico Tecnologico delle Telecomunicazioni
Via Trento, 10
15040 Montecastello (Al.)

Tel. 347 5610956