SOMMARIO

- Programmazione orientata agli eventi.
- Schema di applicazione:
  - `WinMain()` e funzione finestra.
- Gestione tastiera:
  - Output su finestra.
- Gestione mouse:
  - Aggiornamento della finestra.
- Grafica:
  - Punti e linee.
- Risorse.

PROGRAMMAZIONE IN WINDOWS

- Si presentano le *interazioni* fra i programmi e Windows e le *regole* a cui si devono conformare tutte le applicazioni Windows.
- Si illustra lo sviluppo di uno *schema di applicazione* da utilizzare come punto di partenza per altri programmi Windows.
- Windows fornisce un'interfaccia coerente per tutte le applicazioni: la maggior parte del codice che costituisce le applicazioni Windows serve solo a supportare l'interfaccia utente.

PROGRAMMAZIONE IN WINDOWS

- Windows è orientato alla grafica, cioè presenta un'interfaccia utente di tipo grafico (GUI, che sta per *Graphical User Interface*).
- Windows gestisce l'hardware: fornisce ai programmatori l'*indipendenza* dai dispositivi hardware.

PROGRAMMAZIONE IN WINDOWS

- I programmi scritti in modo tradizionale richiedono l'input/output, cioè *effettuano chiamate* al sistema operativo.
- Windows di regola funziona in modo opposto: è *Windows che chiama* il programma utente.
- Le interazioni tra i programmi e il sistema operativo avviene per mezzo dei *messaggi.*
- Un *messaggio* è una notifica che è avvenuto un *evento* (pressione di un tasto, clic del mouse). È un mezzo per smistare le informazioni in un ambiente multitasking a finestre.

PROGRAMMAZIONE IN WINDOWS

- Il programma attende finché non gli viene inviato un messaggio da Windows. Questo messaggio viene passato al programma per mezzo di una funzione speciale *chiamata da Windows*. Quando il messaggio viene ricevuto, si prevede che il programma esegua un'azione appropriata.
- *È l'iterazione con Windows, basata sui messaggi, che determina la forma generale di tutti i programmi Windows.*
- L'arrivo dei messaggi al programma è "casuale".

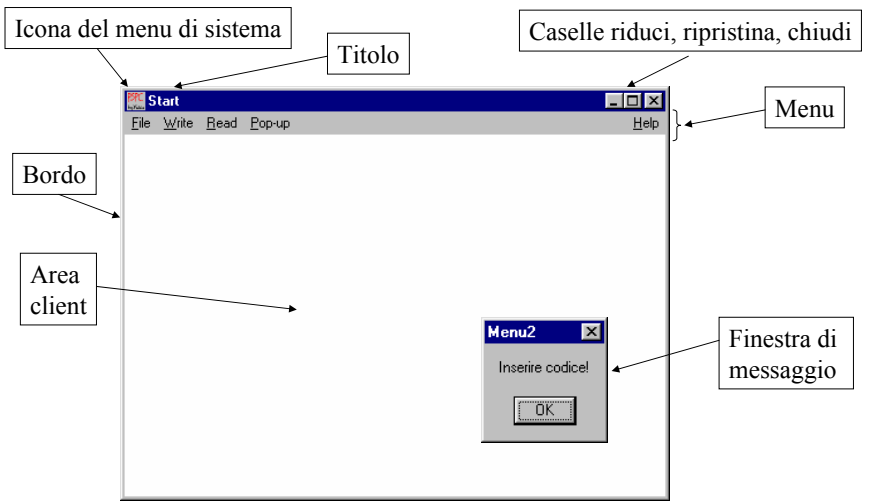
PROGRAMMAZIONE ORIENTATA AGLI EVENTI

- Si può parlare di *programmazione orientata agli eventi*: l'esecuzione del programma è controllata da un flusso di eventi in ingresso all'applicazione.
- Il flusso degli eventi non è proprio dell'applicazione, perché altrimenti ci sarebbero conflitti tra le diverse applicazioni per ottenere l'input dell'utente o l'uso di risorse condivise.
- Per questo motivo il gestore dell'interfaccia grafica (Windows) esegue il controllo degli eventi per conto delle applicazioni, a cui inoltra la notifica degli eventi tramite i messaggi invocando una funzione indicata dall'applicazione.

PROGRAMMAZIONE IN WINDOWS

- All'ambiente Windows si accede per mezzo di un'interfaccia basata su chiamate, detta *Application Program Interface* (API).
- Le funzioni API forniscono tutti i servizi di sistema disponibili in Windows.
- Un sottoinsieme delle API viene chiamato *Graphics Device Interface* (GDI) e costituisce la porzione di Windows che supporta la grafica in modo indipendente dal dispositivo utilizzato.
- Prima di passare ai singoli aspetti della programmazione, definiamo alcuni termini di base.

PROGRAMMAZIONE IN WINDOWS



PROGRAMMAZIONE IN WINDOWS

- Illustriamo alcuni concetti comuni a tutti i programmi Windows.

- **WinMain:** tutti i programmi Windows iniziano l'esecuzione con una chiamata alla funzione `WinMain()`, che ha alcune proprietà speciali che la differenziano dalle altre funzioni dell'applicazione. Deve essere compilata usando la convenzione di chiamata WINAPI e restituisce un intero.

PROGRAMMAZIONE IN WINDOWS

- **Funzione di finestra** (procedura di finestra): è una funzione speciale che non viene chiamata dal programma, ma da Windows. Viene chiamata da Windows quando deve passare un messaggio al programma. Deve restituire il tipo LRESULT CALLBACK. LRESULT è un typedef che corrisponde ad un intero lungo, mentre la convenzione di chiamata CALLBACK viene usata per le *funzioni che sono chiamate da Windows*. Oltre a ricevere i messaggi inviati da Windows la funzione finestra deve avviare tutte le azioni di risposta: il cuore è costituito da un'istruzione *switch* che associa una specifica risposta ad ogni messaggio.

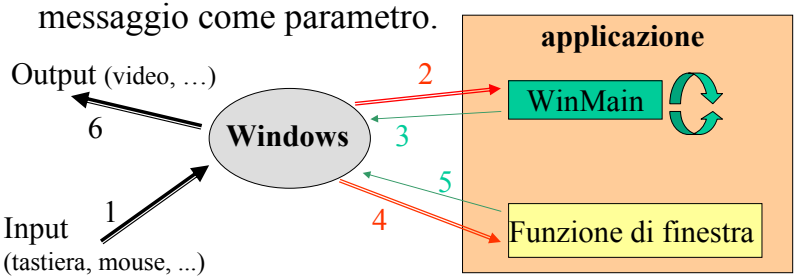
PROGRAMMAZIONE IN WINDOWS

- **Finestra:** quando un programma inizia l'esecuzione deve *definire* e *registrare* una *classe di finestra* (la parola classe non viene usata secondo il significato del C++ o Java, ma come stile o tipo). Registrando una finestra, si indicano a Windows la forma e al funzione della finestra.

- **Handle:** è un valore univoco che identifica vari tipi di risorse come finestre, menu, controlli, icone, allocazioni di memoria.

PROGRAMMAZIONE IN WINDOWS

- **Ciclo dei messaggi:** deve essere definito all'interno della funzione `WinMain()`. Questo ciclo legge i messaggi in attesa dalla *coda messaggi* dell'applicazione e rinvia il messaggio a Windows, che chiama quindi la funzione di finestra del programma utilizzando quel messaggio come parametro.



PROGRAMMAZIONE IN WINDOWS

- **Tipi di dati:** tutti i tipi di dati utilizzati da Windows sono stati definiti con dei `typedef` all'interno del file *windows.h*.

  Alcuni esempi: HWND (intero a 32 bit utilizzato come handle di finestra), BYTE (int a 8 bit), DWORD (int a 32 bit), LPSTR (puntatore a stringa).

  Windows definisce anche numerose strutture, per esempio: MSG (contiene un messaggio) e WNDCLASS (definisce una classe finestra).

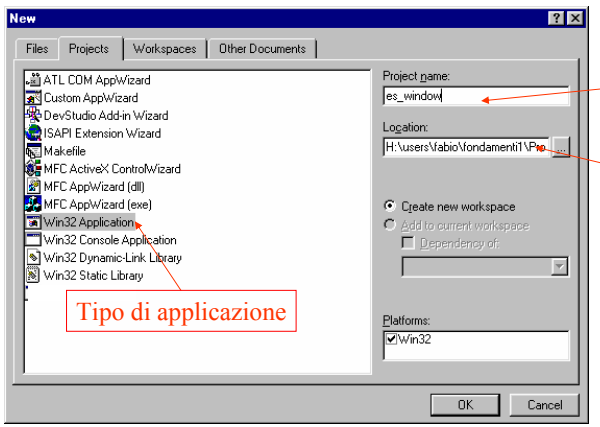
SCHEMA DI APPLICAZIONE

- Si sviluppa uno schema di applicazione Windows con *elementi essenziali* e *comuni* della programmazione Windows (*Win32*). Gli schemi si utilizzano estensivamente in quanto l'infrastruttura di codice necessaria per sviluppare un programma base è molto ampia.

Vediamo uno schema minimo: *creare una finestra ed attenderne la chiusura*. In seguito arricchiremo lo *stesso* codice con nuove funzionalità.

SCHEMA DI APPLICAZIONE

Dal menu *File* del Visual C scegliere *New,* e dal box di dialogo cliccare sul *tab Project.* Dal menu *File* del Visual C scegliere *New,* e dal box di dialogo cliccare sul *tab Files* e scegliere *C++ Source File* con nome my_win.c.



SCHEMA DI APPLICAZIONE

- Un programma Windows minimo contiene due funzioni:
  - **WinMain** che deve:
    - Definire una classe di finestra.
    - Registrare quella classe in Windows.
    - Creare una finestra della classe definita.
    - Visualizzare la finestra.
    - Avviare il ciclo di messaggi.
  - **Funzione finestra** che deve:
    - Rispondere a tutti i messaggi rilevanti.

SCHEMA DI APPLICAZIONE

```
/*Uno schema minimo di programma Windows*/
#include <windows.h>
```

Contiene i prototipi delle API, oltre a vari tipi, macro e definizioni

Dichiarazione della funzione finestra

```
LRESULT CALLBACK WndFunc (HWND, UINT, WPARAM, LPARAM);
```

```
int WINAPI WinMain (...)
{
    …istruzioni…
}
```

Creazione della finestra e avvio del ciclo dei messaggi

```
LRESULT CALLBACK WndFunc(...)
{
…istruzioni…
}
```

Gestione dei messaggi

WinMain()

```
int WINAPI WinMain (HINSTANCE hThisInst, HINSTANCE hPrevInst,
                    LPSTR lpszArgs, int nWinMode)
```

- L'handle `hThisInst` fa riferimento alla sessione di programma corrente (istanza).
- `hPrevInst` è sempre NULL.
- `lpszArgs` è un puntatore a una stringa che contiene gli eventuali argomenti della riga di comando.
- `nWinMode` descrive il modo in cui la finestra viene visualizzata quando il programma inizia l'esecuzione.

WinMain()

- Al suo interno dichiariamo tre variabili:

```
int WINAPI WinMain (…
{
      HWND        hwnd ;
      MSG         msg ;
      WNDCLASS    wcl ;
…
}
```

Contiene l'handle della finestra del programma

La *struttura* `msg` contiene i messaggi della finestra

La *struttura* `wcl` viene utilizzata per definire la classe di finestra

DEFINIZIONE DELLA CLASSE FINESTRA

• Per definire una classe si riempiono i campi della *struttura* WNDCLASS.

Stile della finestra: normale

```
…
wcl.style         = 0 ;
wcl.lpfnWndProc   = WndFunc ;
wcl.cbClsExtra    = 0 ;
wcl.cbWndExtra    = 0 ;
wcl.hInstance     = hThisInst ;
wcl.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
wcl.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
wcl.hbrBackground = GetStockObject (WHITE_BRUSH) ;
wcl.lpszMenuName  = NULL ;
wcl.lpszClassName = "MyWin" ;
…
```

Informazioni su allocazione

Nessun menu

Nome della classe finestra

DEFINIZIONE DELLA CLASSE FINESTRA

- Al campo `hInstance` viene assegnato l'handle dell'istanza corrente.
- Al campo `lpfnWndProc` è assegnato l'indirizzo della funzione finestra.

- Per ogni applicazione si deve definire una forma per il cursore del mouse e per l'icona, ogni applicazione può definire una propria versione di tali risorse.

DEFINIZIONE DELLA CLASSE FINESTRA

- Lo stile dell'icona viene caricato dalla funzione API `LoadIcon ()`: restituisce un handle di icona e richiede come parametri l'handle del modulo che contiene l'icona e il nome dell'icona.

- Lo stile del cursore viene caricato dalla funzione API `LoadCursor()`: restituisce un handle di risorsa cursore e richiede come parametri l'handle del modulo che contiene il cursore e il nome del cursore.

DEFINIZIONE DELLA CLASSE FINESTRA

- La funzione API `GetStockObject()` definisce il colore dello sfondo della finestra: restituisce un handle che identifica la risorsa specificata come parametro (si ottiene un handle GDI). In questo caso si considera un *pennello*, che è una risorsa che colora lo schermo:

```
HGDIOBJ  GetStockObject (int  object);
```

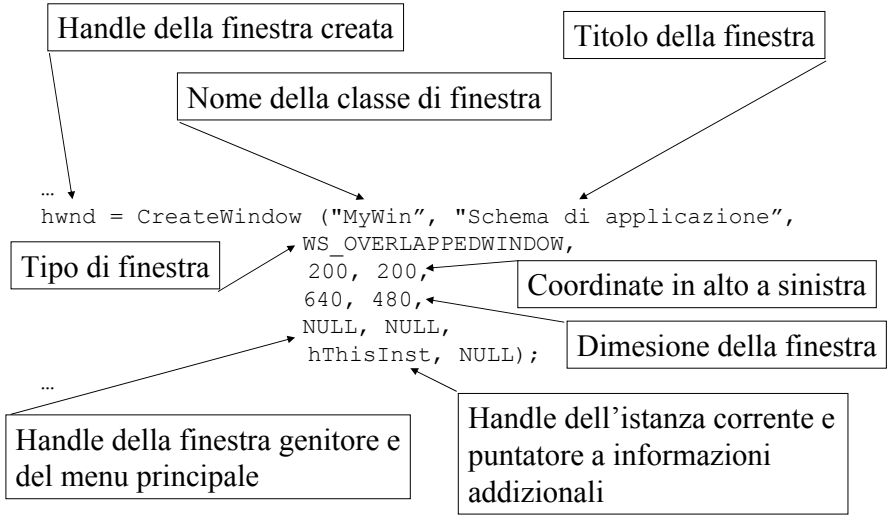

REGISTRAZIONE DELLA FINESTRA

- Completata la definizione della classe di finestra, la si registra in Windows:

```
…
RegisterClass (&wcl) ;
…
```

- Si crea una finestra appartenente alla classe appena definita usando la funzione API `CreateWindow()`.

CREAZIONE DI UNA FINESTRA



VISUALIZZAZIONE DI UNA FINESTRA

- Per visualizzare la finestra si chiama la funzione API `ShowWindow()`. Occorre anche utilizzare la funzione API `UpdateWindow()` che indica a Windows di inviare un messaggio all'applicazione quando è necessario aggiornare la finestra:

```
…
ShowWindow (hwnd, nWinMode) ;
UpdateWindow (hwnd) ;
…
```


CICLO DEI MESSAGGI

- La parte finale di `WinMain()` è costituito dal *ciclo dei messaggi*.

```
...
while (GetMessage (&msg, NULL, 0, 0))
        {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
        }
return msg.wParam ;
}
```

- `GetMessage()`, appena l'applicazione è pronta, legge i messaggi inviati da Windows nella coda dei messaggi e se non vi sono messaggi passa il controllo a Windows. Ritorna *zero* quando l'utente termina il programma.

CICLO DEI MESSAGGI

```
BOOL GetMessage(LPMSG msg, HWND nwnd,
                UINT min,UINT max);
```

- Quando vi è un messaggio, la funzione API `TranslateMessage()` lo elabora, infine `DispatchMessage()` lo rinvia a Windows che lo trattiene finché non può inviarlo alla funzione finestra del programma.

- Quando il ciclo dei messaggi termina la funzione `WinMain()` restituisce il codice di ritorno, contenuto in `msg.wParam`, all'ambiente (Windows).

CICLO DEI MESSAGGI

- Tutti i messaggi hanno il tipo di struttura `MSG`, che è definita nei file header di Windows:

```
typedef struct tagMSG {
  HWND   hwnd;     /*finestra a cui è destinato il messaggio*/
  UINT   message;  /*il "tipo" di messaggio*/
  WPARAM wParam;   /*informazioni dipendenti dal messaggio*/
  LPARAM lParam;   /*altre informazioni dal messaggio*/
  DWORD  time;     /*orario di invio del messaggio*/
  POINT  pt;       /*coordinate del mouse*/
} MSG;
```

```
typedef struct tagPOINT
{
 LONG x; LONG y;
 } POINT;
```


WINMAIN

- Quindi la funzione `WinMain()` completa è:

```
int WINAPI WinMain (HINSTANCE hThisInst, HINSTANCE hPrevInst,
                    LPSTR lpszArgs, int nWinMode)
{
     HWND        hwnd ;
     MSG         msg ;
     WNDCLASS    wcl ;

     wcl.style         = 0 ;
     wcl.lpfnWndProc   = WndFunc ;
     wcl.cbClsExtra    = 0 ;
     wcl.cbWndExtra    = 0 ;
     wcl.hInstance     = hThisInst ;
     wcl.hIcon         = LoadIcon (NULL,IDI_APPLICATION) ;
     wcl.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
     wcl.hbrBackground = GetStockObject (WHITE_BRUSH) ;
     wcl.lpszMenuName  = NULL ;
     wcl.lpszClassName = "MyWin" ;
                                                    . . .
```


WINMAIN

```
. . .
    RegisterClass (&wcl) ;

    hwnd = CreateWindow ("MyWin", "Schema di applicazione",
                         WS_OVERLAPPEDWINDOW,
                          50, 50,
                         640, 480,
                         NULL, NULL, hThisInst, NULL) ;

    ShowWindow (hwnd, nWinMode) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
         {
            TranslateMessage (&msg) ;
            DispatchMessage (&msg) ;
         }
    return msg.wParam ;
}
```


LA FUNZIONE FINESTRA

- La seconda funzione nello schema di applicazione è la funzione finestra: `WndFunc()` (può avere qualsiasi nome):

```
LRESULT CALLBACK WndFunc(HWND hwnd,UINT message,
                         WPARAM wParam,LPARAM lParam)
{
     switch(message){
     case WM_DESTROY:
        PostQuitMessage(0);
        break;
     default:
        return DefWindowProc(hwnd, message, wParam, lParam);
     }
     return 0;
}
```

LA FUNZIONE FINESTRA

- La funzione finestra deve gestire le azioni in risposta ai messaggi: infatti alla funzione finestra vengono passati come parametri i primi quattro membri della struttura `MSG`.
- Il corpo della funzione è costituito da uno `switch` sul "tipo" di messaggio: in questo caso si risponde esplicitamente ad un unico messaggio `WM_DESTROY`. Tale messaggio è inviato quando l'utente termina il programma.

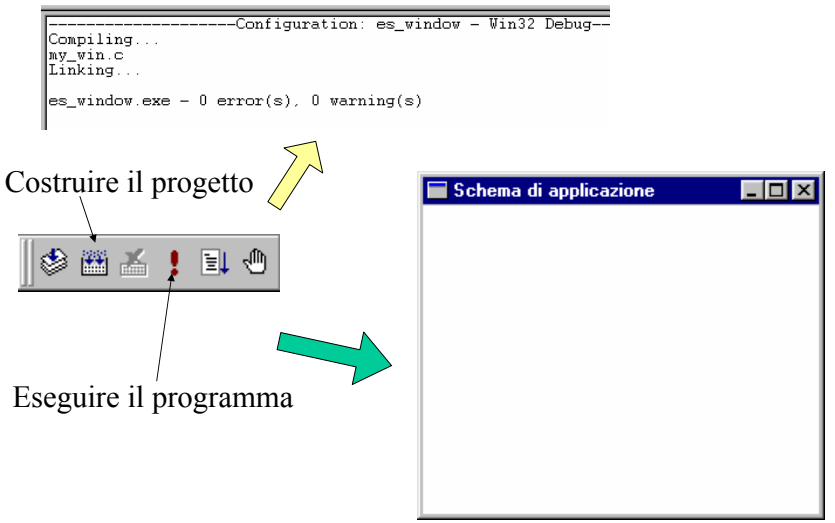
LA FUNZIONE FINESTRA

- Per rispondere al messaggio di chiusura si deve utilizzare la funzione API `PostQuitMessage()` che ha come argomento un codice di uscita che viene restituito in `msg.wParam` all'interno di `WinMain()`.
- Chiamando tale funzione si fa in modo che venga inviato un messaggio `WM_QUIT` all'applicazione, quindi *`GetMessage()`* restituisce falso e interrompe il ciclo dei messaggi.
- Qualsiasi altro messaggio deve essere ripassato a Windows: quindi si chiama `DefWindowProc()` per l'elaborazione standard.

CONVENZIONI SUI NOMI

- La convenzione sui nomi è stata coniata da Microsoft appositamente per Windows.
- Per le funzioni il nome consiste in un verbo seguito da un nome: il primo carattere del verbo e del nome sono maiuscoli.
- Per i nomi delle variabili si usa un prefisso in caratteri minuscoli per inserire il tipo dei dati nel nome. Il nome vero e proprio inizia con un carattere maiuscolo.
  - *c* carattere, *fn* funzione, *h* handle, *p* puntatore, *n* intero, *sz* puntatore a stringa.

ESECUZIONE DELL'ESEMPIO



ELABORAZIONE DEI MESSAGGI

- Si amplia lo schema di applicazione di base in modo da poter ricevere ed elaborare i messaggi più comuni.
- In Windows ogni *messaggio* è rappresentato da un intero a 32 bit *univoco* e viene utilizzato attraverso delle macro (come quella vista `WM_DESTROY`).
- Ad ogni messaggio sono associati altri due valori specifici di tipo `WPARAM` e `LPARAM` che sono chiamati `wParam` e `lParam`: contengono informazioni relative alle coordinate del mouse, al tasto premuto o a dati di sistema.

ELABORAZIONE DEI MESSAGGI

- Tali informazioni vengono passate come parametri alla funzione di finestra.
- Talvolta nelle variabili `wParam` e `lParam` sono contenute due informazioni: per facilitare l'accesso a ciascuna metà Windows definisce due *macro* chiamate `LOWORD` e `HIWORD`. Esse restituisco la parte inferiore e superiore del contenuto della variabile.

PRESSIONE DI UN TASTO

Il modo più comune per interagire con un programma è premere dei tasti: modifichiamo lo schema di applicazione per fare in modo che alla pressione del tasto "c" scriva sulla finestra "Ciao, Mondo".

- In Windows l'applicazione non riceve direttamente i caratteri battuti dalla tastiera, ma riceve un messaggio `WM_CHAR` che indica che è avvenuta la pressione di un tasto: il parametro `wParam` contiene il valore ASCII del tasto premuto.

PRESSIONE DI UN TASTO

```
…
int WINAPI WinMain (…
hwnd = CreateWindow ("MyWin", "Schema di
                              applicazione: pressione tasto",…
…
LRESULT CALLBACK WndFunc(HWND hwnd,UINT message,
                         WPARAM wParam,LPARAM lParam){
       HDC hdc;
       char *str="Ciao, Mondo";

       switch(message){
       case WM_CHAR:
              hdc=GetDC(hwnd);
              if ((char)wParam=='c')
                     TextOut(hdc,20,20,str,strlen(str));
              ReleaseDC(hwnd,hdc);
              break;
       case WM_DESTROY:
             …
}
```

PRESSIONE DI UN TASTO

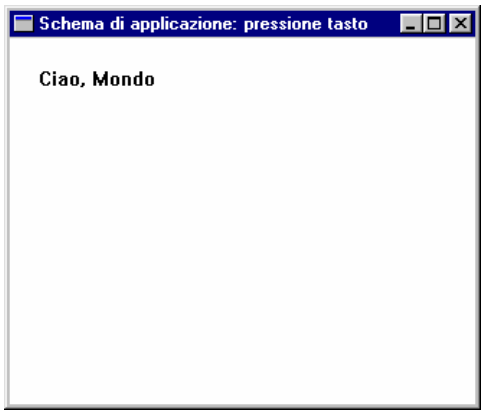
- In `WinMain()` si modifica il titolo della finestra.
- Nella funzione finestra si aggiunge un `case` nello `switch` per la gestione del nuovo messaggio.
- Si definisce una stringa nella funzione finestra.
- Tuttavia non è possibile scrivere direttamente sulla finestra: Windows deve creare un collegamento tra il programma e lo schermo. Questo viene chiamato contesto di dispositivo (*Device Context*).

CONTESTO DI DISPOSITIVO

- Il contesto di dispositivo è un percorso di output che va dall'applicazione all'*area client* dell'applicazione tramite *il driver di dispositivo appropriato.*
- Il collegamento si ottiene con `GetDC()` che restituisce un handle di contesto di dispositivo relativo alla finestra passata come argomento.
- Essendo una risorsa limitata si deve rilasciare il contesto di dispositivo al termine del suo utilizzo tramite `ReleaseDC()`.

OUTPUT SU MONITOR

- Per scrivere sulla finestra si usa `TextOut()`: i suoi argomento sono il contesto di dispositivo, la posizione in pixel del testo, la stringa del testo e la lunghezza della stringa.



GESTIONE DEL MOUSE

- Windows è un sistema operativo basato sull'uso del mouse, pertanto vi sono molti messaggi correlati al mouse. Noi ne considereremo uno: la pressione del *tasto sinistro*.

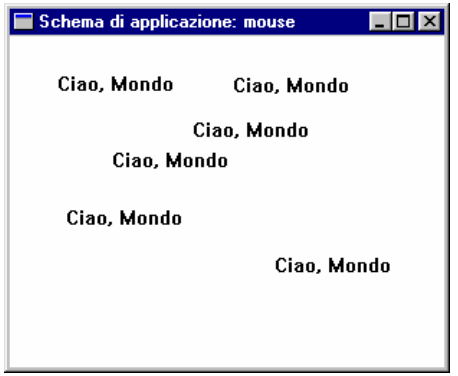
Modifichiamo lo schema di applicazione per fare in modo che la stringa "Ciao, Mondo" venga scritta nella posizione del mouse quando si clicca il pulsante sinistro.

GESTIONE DEL MOUSE

```
case WM_LBUTTONDOWN:
   hdc=GetDC(hwnd);
   TextOut(hdc,LOWORD(lParam),HIWORD(lParam),str,strlen(str));
   ReleaseDC(hwnd,hdc);
   break;
```

- `WM_LBUTTONDOWN` è la macro che identifica la pressione del tasto sinistro del mouse.
- Le coordinate della posizione corrente del mouse sono contenute in `lParam`: la *coordinata* x nella parte bassa della variabile e la y nella parte alta.

GESTIONE DEL MOUSE



Nell'uso dell'applicazione si nota che: (1) quando la finestra viene coperta da altre finestra si perde il contenuto; (2) la stringa "Ciao, Mondo" non si cancella tra un clic e il successivo.

Questi fatti sono tutti legati all'*aggiornamento della finestra* e alla sua gestione.

AGGIORNAMENTO DELLA FINESTRA

- Non è compito di Windows tenere traccia del contenuto delle finestre, è il programma che mantiene il contenuto della finestra (è l'applicazione che crea il contenuto).
- Ogni volta che il contenuto della finestra deve essere visualizzato di nuovo al programma viene inviato il messaggio `WM_PAINT` (tale messaggio viene inviato anche al momento in cui la finestra viene visualizzata per la prima volta).
- Quindi si deve gestire questo messaggio nello `switch`.

AGGIORNAMENTO DELLA FINESTRA

- Dato che il messaggio `WM_PAINT` implica ridisegnare l'area client è buona norma inserire in questo `case` le istruzioni relative alla grafica.
- Anche in questo caso si deve creare un contesto di dispositivo, tuttavia si utilizzano una funzioni API diverse (si devono gestire, in generale, situazioni più complesse): `BeginPaint()` e `EndPaint()`.

AGGIORNAMENTO DELLA FINESTRA

```
…
PAINTSTRUCT paintstruct;
…
        case WM_PAINT:
               hdc=BeginPaint(hwnd,&paintstruct);
               TextOut(hdc,20,20,"Uso WM_PAINT",12);
               EndPaint(hwnd,&paintstruct);
               break;
…
```

- Il secondo parametro è un puntatore ad una struttura che contiene alcune informazioni ed in particolare le coordinate dell'area che deve essere ridisegnata.

AGGIORNAMENTO DELLA FINESTRA

- Per completare l'applicazione relativa al mouse si può trasferire la parte di visualizzazione al *case* relativo al messaggio `WM_PAINT`. *Tuttavia si deve tenere presente che tra i due messaggi* (`WM_LBUTTONDOWN` e `WM_PAINT`) *si esce dalla funzione di finestra*: quindi le coordinate si devono memorizzare in variabili globali.
- Inoltre per far in modo che la visualizzazione avvenga al clic del mouse (e non solo quando la finestra viene coperta) si deve chiedere a Windows di inviare un messaggio `WM_PAINT`.

AGGIORNAMENTO DELLA FINESTRA

- Quindi è possibile che l'applicazione provochi la generazione di un messaggio `WM_PAINT`: non si ridisegna l'area client "direttamente" per lasciare decidere a Windows quando è il momento più opportuno (l'aggiornamento è un compito dispendioso).
- Si deve chiamare la funzione API `InvalidateRect()`: il primo parametro è l'handle della finestra; un puntatore ad una struttura `RECT` che contiene le coordinate dell'area rettangolare da aggiornare e un valore booleano per indicare se cancellare lo sfondo.

AGGIORNAMENTO DELLA FINESTRA

```
…
#include <stdio.h>
…
int x=1,y=1;
char str[64]=" ";

LRESULT CALLBACK WndFunc(…
…
case WM_PAINT:
       hdc=BeginPaint(hwnd,&paintstruct);
       TextOut(hdc,x,y,str,strlen(str));
       EndPaint(hwnd,&paintstruct);
       break;
case WM_LBUTTONDOWN:
       x=LOWORD(lParam);
       y=HIWORD(lParam);
       sprintf(str,"Tasto sinistro: (%d,%d)",x,y);
       InvalidateRect(hwnd,NULL,1);
       break;
…
```

Serve per `sprintf()`

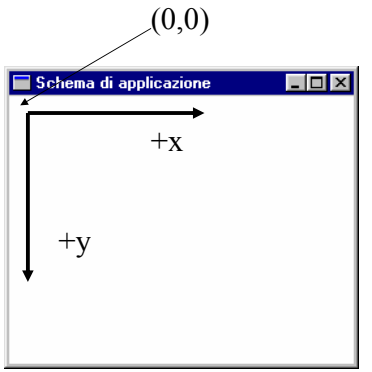
Variabili globali. Inizializzazioni utili per il primo `WM_PAINT`

AGGIORNAMENTO DELLA FINESTRA

GRAFICA

A questo punto si può considerare un'applicazione che generi output grafico: disegnare un piano cartesiano con punti, segmenti, rettangoli, cerchi.

- Per prima cosa è importante sapere come sono considerate le coordinate di una finestra: il punto (0,0) è in alto a sinistra, l'asse y è verticale e rovesciato.



GRAFICA

- Per disegnare sono sufficienti due funzioni: una per scrivere un pixel e l'altra per leggerlo. Quindi si possono costruire le figure per punti. Tuttavia questa tecnica è molto onerosa: infatti di solito si usano funzioni che disegnano figure più complesse (per esempio usando le routine ottimizzate dei driver della scheda video).

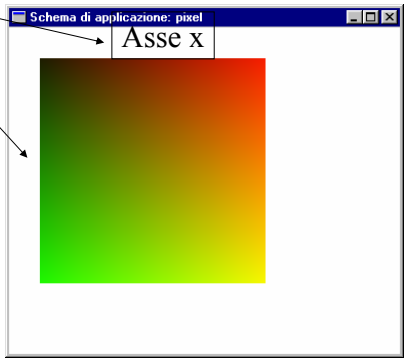
- Consideriamo prima le funzioni GDI per disegnare punti e linee.

GRAFICA: PUNTI

- Per tracciare un pixel si usa la funzione `SetPixel(hdc, x, y, rgbColor)`.
- I colori sono forniti attraverso la macro `RGB()` che ha come argomenti le *intensità* (espresse in piccoli interi [0,255]) dei tre colori fondamentali (*rosso, verde e blu*).
- Per esempio, `SetPixel(hdc, 10, 10, RGB(0,200,0))` visualizza un punto di colore verde intenso nella posizione (10,10).

GRAFICA: PUNTI

```
…
case WM_PAINT:
  hdc=BeginPaint(hwnd,&paintstruct);
  for (i=30;i<250;i++)
    for(j=30;j<250;j++)
      SetPixel(hdc,i,j,RGB(i,j,0));
  EndPaint(hwnd,&paintstruct);
  break;
…
```



Il codice descrive una tabella bidimensionale (matrice) i cui valori crescono come gli indici: si ottengono diverse combinazioni di colori variando le intensità del rosso e del verde.

GRAFICA: LINEE

- Per disegnare linee si utilizza la funzione `LineTo(hdc, xEnd, yEnd)` disegna una linea dalla posizione corrente (il valore di default è (0,0)) alla posizione (*xEnd, yEnd)*.
- Per spostare la posizione corrente si usa la funzione `MoveToEx(hdc, xStart, yStart, lpPoint)` che sposta il riferimento in (*xStart, yStart*) (nel puntatore *lpPoint* viene inserita la vecchia posizione).
- Lo stile della linea dipende dalla "penna" corrente: ne esistono tre predefinite, ma se ne possono creare di personalizzate.

GRAFICA: LINEE

- Per ottenere un handle di penna si utilizza:

```
HPEN hPen;
hPen = GetStockObject(BLACK_PEN);
```

- Per rendere corrente la penna nel contesto di dipositivo si usa:

```
SelectObject(hdc,hPen);
```


GRAFICA: FIGURE GEOMETRICHE

- Alcune funzioni Windows predisposte alla creazione di figure sono: `Rectangle()`, `Ellipse()`, `RoudRect()`, ve ne sono altre per disegnare archi, settori e poligoni.
- Windows traccia il contorno con la penna selezionata e la figura viene riempita con il "pennello" corrente (ci sono sei pennelli a disposizione, ma se ne possono creare di personalizzati).

GRAFICA: FIGURE GEOMETRICHE

- Per ottenere un handle di pennello si utilizza:

```
HBRUSH hBrush;
hBrush = GetStockObject(GRAY_BRUSH);
```

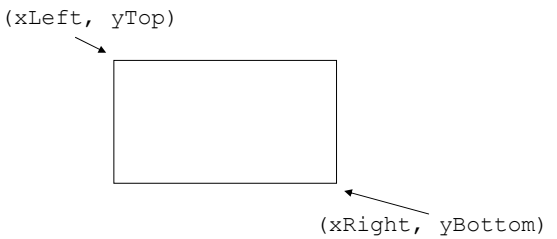
- Per rendere corrente il pennello nel contesto di dipositivo si usa:

```
SelectObject(hdc,hBrush);
```


GRAFICA: FIGURE GEOMETRICHE

- Le funzioni viste generano le figure geometriche in base al rettangolo circoscritto:

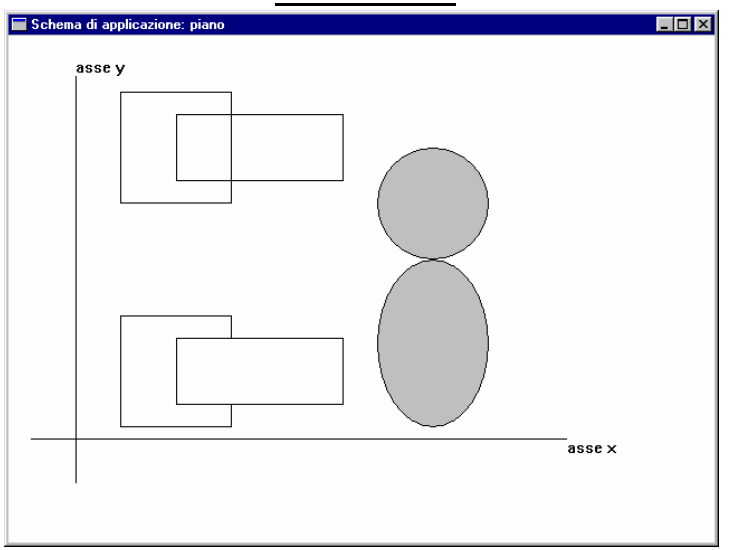
```
Rectangle(hdc, xLeft, yTop, xRight, yBottom);
```



ESEMPIO

```
…
case WM_PAINT:
       hdc=BeginPaint(hwnd,&paintstruct);
       MoveToEx(hdc,60,20,NULL);LineTo(hdc,60,400);
       TextOut(hdc,60,20,"asse y",6);
       MoveToEx(hdc,20,360,NULL);LineTo(hdc,500,360);
       TextOut(hdc,500,360,"asse x",6);
       Rectangle(hdc,100,250,200,350);
       Rectangle(hdc,150,270,300,330);
       SelectObject(hdc,GetStockObject(NULL_BRUSH));
       Rectangle(hdc,100,50,200,150);
       Rectangle(hdc,150,70,300,130);
       SelectObject(hdc,GetStockObject(LTGRAY_BRUSH));
       Ellipse(hdc,330,100,430,200);
       Ellipse(hdc,330,200,430,350);
       EndPaint(hwnd,&paintstruct);
       break;
…
```


ESEMPIO



INTERFACCIA UTENTE

- Si è considerato come lo schema base di applicazione riceve ed elabora i messaggi, vediamo adesso i componenti che costituiscono l'*interfaccia utente* di Windows.
- Un'applicazione Windows comunica con l'utente attraverso uno o più elementi di interfaccia predefiniti: *icone, finestre di messaggio, menu e finestre di dialogo.*
- Questi elementi di interfaccia vengono chiamati *risorse.*

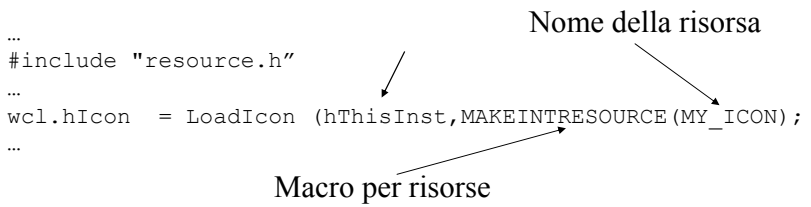
INTERFACCIA UTENTE

- Le *risorse* sono oggetti utilizzati dal programma ma definiti al di fuori di esso: le risorse sono create separatamente rispetto al programma ma sono aggiunte al file eseguibile quando se ne esegue il link.
- Le *risorse* sono contenute in file risorsa, con estensione *.rc,* e sono create con editor appositi. Tale file non è scritto in C, ma in un linguaggio speciale per le risorse, che deve essere compilato (in Visual C tali passi sono fatti in modo automatico, si deve solo includere il file *resource.h*).

ICONE

Consideriamo di modificare lo schema di applicazione in modo tale che il file eseguibile sia rappresentato con un'icona.

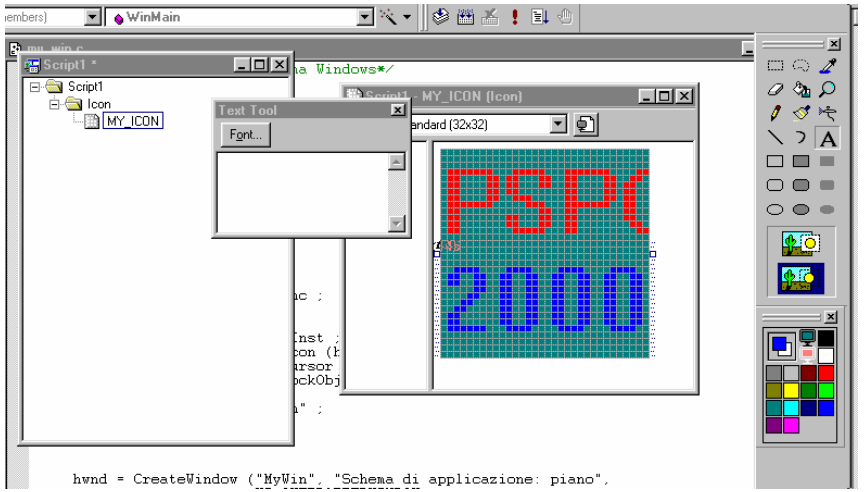
- Si deve includere il file `resource.h` che è generato dall'ambiente di sviluppo e caricare la risorsa con la funzione `LoadIcon()`:



ICONE

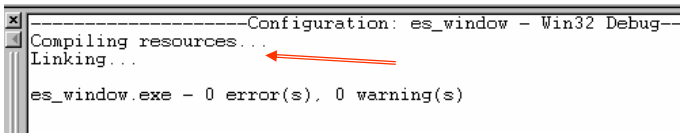
- Dal menu *Insert* del Visual C scegliere *Resource,* e dal box di dialogo cliccare sul *Icon* (visualizzata come icona).
- Si entra nell'editor di risorsa per manipolare le icone, che è simile ad un programma di grafica con i relativi strumenti.

ICONE



ICONE

- Si aggiunge il file .rc al progetto (tasto destro del mouse sul *tab File View* della finestra workspace) e quindi si costruisce l'eseguibile.



Creando un collegamento al file eseguibile sul desktop si ottiene la rappresentazione a icona.

FINESTRE DI MESSAGGIO

- Una finestra di messaggio visualizza un messaggio per l'utente e attende una conferma. Si può utilizzare la finestra di messaggio per scegliere tra alcune alternative.
- Per creare la finestra di messaggio si usa la funzione `MessageBox()` il primo argomento è l'handle della finestra dell'applicazione, seguito dalla stringa da visualizzare, dal titolo della finestra di messaggio ed infine dal tipo (diverse forme e numero di pulsanti). Ritorna un intero che rappresenta la scelta fatta.

FINESTRE DI MESSAGGIO

- La gestione della finestra di messaggio è automatica, data la sua semplicità può essere utile nella fase di debug inserendola in modo opportuno (simile a `printf()`) nel codice

Modifichiamo lo schema di applicazione della pressione di un carattere per fare in modo che venga visualizzata una finestra di messaggio con relativa scelta.

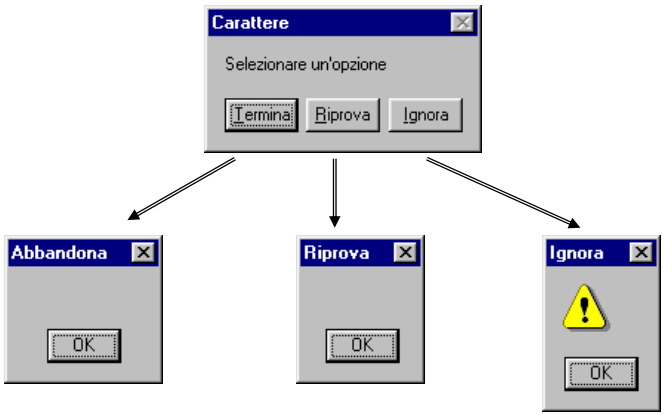
FINESTRE DI MESSAGGIO

```
int res;
…
case WM_CHAR:
       res=MessageBox(hwnd,"Selezionare un'opzione",
                      "Carattere",MB_ABORTRETRYIGNORE);
       switch(res){
       case IDABORT:
              MessageBox(hwnd," ","Abbandona",MB_OK);
              break;
       case IDRETRY:
              MessageBox(hwnd," ","Riprova",MB_OK);
              break;
       case IDIGNORE:
              MessageBox(hwnd," ","Ignora",
                             MB_OK|MB_ICONEXCLAMATION);
              break;
       }
       break;
…
```

Macro

FINESTRE DI MESSAGGIO

- Quando si preme un tasto sull'area client si ottiene:



INTRODUZIONE AI MENU

- In Windows l'elemento di controllo più comune è il *menu*: qualsiasi finestra principale ha un menu associato.
- L'aggiunta di un *menu* a una finestra comprende alcuni passi:
  - Definire la forma del *menu* in un file risorsa (per mezzo di un tool fornito dall'IDE).
  - Caricare il *menu* alla creazione delle finestra.
  - Elaborare le selezioni da *menu.*

INTRODUZIONE AI MENU

- In Windows il livello più alto di menu è visualizzato sulla riga superiore della finestra. I sottomenu appaiono come menu a comparsa (*popup*).
- Una volta creato (disegnato) un menu, si include nel programma specificandone il nome quando si crea la classe di finestra:

```
wcl.lpszMenuName  = MAKEINTRESOURCE(MY_MENU);
```

Macro per risorse

Nome risorsa

INTRODUZIONE AI MENU

- Ogni volta che un utente seleziona un menu, alla funzione finestra viene inviato il messaggio `WM_COMMAND`. Il valore `LOWORD(wParam)` corrisponde all'identificatore di quel menu (cioè il valore associato a quell'elemento di menu durante la sua definizione). Pertanto è necessario usare un'istruzione `switch` annidata per determinare l'elemento selezionato.