1. Noțiuni elementare de WinApi

1.1. Introducere - cum arată un program de windows?

Un program de windows este alcătuit dintr-o fereastră principal cu o structură standard (de obicei). Această fereastră se află pe ecran pe parcursul întregii desfăşurări a aplicației, aşteptând input de la utilizator/system la care reacționează în mod adecvat..

Aplicațiile windows sunt aplicații care au la bază *evenimente* (*event driven*), de exemplu de la mouse, menu, tastatură, butoane etc.

Evenimentele generează *mesaje*, care sunt plasate într-o *coadă de mesaje*, urmând a fi procesate secvențial. Coada de mesaje aşteaptă mesaje pe parcursul întregii durate a aplicației

1.2. Structura de bază a unui program windows

Un program windows conține cel puțin două funcții de bază:

```
(1) int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    PSTR szCmdLine, int iCmdShow)
(2) LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

(1) Funcția `WinMain` (echivalentă cu `void main()` in C):

- Creează o clasă window pentru fereastra principal şi înregistează această clasă.
- Creează o nouă fereastră ca instanță a acestei clasei window.
- Afişază fereastra creată pe ecran.
- Lansează bucla de mesaje

Observații:

- Fiecare fereastră are un ***handle*** (identificator de fereastră) pe baza căreia poate fi referențiată
- Fiecare fereastră aparține unei instanțe a programului
- Evenimentele unei ferestre sunt gestionate de către procedura de ferastră

(2) Funcția WndProc (poate avea orice nume! Nu doar WndProc):

- Reprezintă procedura de fereastră
- Determină răspunsul la diferitele mesaje generate de evenimente

Exemplu: minapp.cpp

```
#include <windows.h>
#include "definitions.h"      // Include definitions
#include "FrameGrabber.h"     // Include FrameGrabber class
#include "resource.h"         // Include resources
#include "loadsave.h"         // Include headers of load and save functions
#include "result.h"           // Include hader for result window

//
// Prototipe of the window procedure, which handles the events
//
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);


FrameGrabber  *fg;                  // FrameGrabber object
bool  passthruOn;                   // Only for image aquisition
HWND  hwndmain;                     // Handle of the main window

HINSTANCE myinstance;               //Current instrance

BYTE *buffer1,*buffer2;             // Buffers for the image pixels

//
// Main function
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                           PSTR szCmdLine, int iCmdShow)
{
      myinstance=hInstance;
      WNDCLASS wcmain;        // Structure of window class
      MSG   msg;              // Variable for messages
      char * szWindowTitle   = "Digital Image Processing"; //Window title
      char * szClassName    = "dbvapp"; //Window class name
      int xPos = 100, yPos = 100;   //Coordinates of main window on screen
      int  width = MAX_COL+20 ,height = MAX_ROW + 50; //width and height of
                                                      //main window

      buffer1 = new BYTE[ALL_PIXELS];     //Pixel buffers for image pixels

      buffer2 = new BYTE[ALL_PIXELS];

      // Test if there was a previous instance of the application
      if (hPrevInstance == 0)
      {
            wcmain.style = CS_BYTEALIGNWINDOW|CS_HREDRAW|CS_VREDRAW|
                           CS_DBLCLKS; //Window stiles
            wcmain.lpfnWndProc = WndProc; //Sets the window procedure
            wcmain.cbClsExtra   = 0;
            wcmain.cbWndExtra   = 0;
            wcmain.hInstance    = hInstance; //Sets the instance
            wcmain.hIcon        = LoadIcon(NULL, IDI_APPLICATION);
            wcmain.hCursor      = LoadCursor(NULL, IDC_ARROW);
            wcmain.hbrBackground= (HBRUSH) GetStockObject(BLACK_BRUSH);
            wcmain.lpszMenuName = MAKEINTRESOURCE(IDR_DBVAPP); //Sets menu
            wcmain.lpszClassName= szClassName; //Sets name foe window class

            //Each new class has to be registered
            if (!RegisterClass(&wcmain))
            {
```

```
                MessageBox(NULL,TEXT("Programm works with Unicode and
                requires Windows NT!"),szClassName, MB_ICONERROR);
                return 0;
            }
        }

// Create main window. Returns as result the handle to the window!
   hwndmain = CreateWindow(
                szClassName,       // Name of the corresponding class
                szWindowTitle,     //String which will appear in the title
                                   //bar
            WS_SYSMENU|WS_CAPTION|WS_OVERLAPPED|WS_BORDER|WS_MINIMIZEBOX,
            //stiles of the window
               xPos, yPos,         // Coordinates of upper left corner of
                                   //the window on the screen
               width, height,      // Dimensions of the window
               NULL,               // handle to parent Window (none)
               LoadMenu(hInstance, MAKEINTRESOURCE(IDR_DBVAPP)), //Menu
               hInstance,          // Instance to which it belongs
               NULL);
   // Test if the window was created successfully
   if (hwndmain == NULL)
   {
          MessageBox(NULL, TEXT("Window could not be opened!"),
                              szClassName, MB_ICONERROR);
          return 0;
   }


 // Initialize the Framegrabber (instance of the FrameGrabber class)
   if (!(fg = new FrameGrabber("dbv")))
   {
          MessageBox(NULL, TEXT("Framegrabber could not be initialized!"),
                  szClassName, MB_ICONERROR);
          return 0;
   }

   // Create to frames for the FrameGrabber object
   fg->NewFrame();
   fg->NewFrame();

     // Show and update created window
   ShowWindow(hwndmain, iCmdShow);
   UpdateWindow(hwndmain);

     // Message queue
   while (GetMessage(&msg, NULL, 0, 0))
   {
          TranslateMessage(&msg);
          DispatchMessage(&msg);
   }

   return msg.wParam;
}

//Window Procedure. Has as parameters the handle of the window to which
//it belongs, the message received and suplimentar information about the
//message
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
   switch (message)
   {
```

```
   // message generated when creating the window
   case WM_CREATE:
          return 0;

   // medssage genrated when destroying the window
   case WM_DESTROY:
          delete fg;
          PostQuitMessage(0); // ends the program by sending WM_QUIT message
          return 0;

   // message generated when drawing or redrawing the window
   case WM_PAINT:
         HDC      hdc; //handle to a device context
         PAINTSTRUCT   ps;
         hdc = BeginPaint(hwnd, &ps); //painting has allways to begin with
                                      //this function
         fg->DrawFrame(1,hwnd); //draws the pixels saved in frame 1 in the
                                //window with the handle hwnd
         EndPaint(hwnd, &ps); //painting has to end with this function
         return 0;
   // wenn Menuepunkte angewaelt werden
   case WM_COMMAND:
      switch(LOWORD(wParam))
      {
        case ID_OPEN:
              LoadImage(hwnd);
              break;
        case ID_SAVE:
              SaveImage(hwnd);
              break;

      case ID_CLOSE:
           // Verschickt die WM_DESTROY Nachricht an das Fenster hwnd
             DestroyWindow(hwnd);
         return 0;
      default:  // Nachrichten weiterleiten
             return DefWindowProc(hwnd, message, wParam, lParam);
      }

       default:    // Nachrichten weiterleiten
          return DefWindowProc(hwnd, message, wParam, lParam);

   }
   return 0;
}
```

1.3. Explicitarea structurii programului principal

Funcția WinMain

(1) Explicitatea parametrilor

Funcția WinMain are 4 parametri:

- *hInstance* (instance handle) :
  - reprezintă un număr care identifică în mod unic programul aflat în execuție (este posibil ca utilizatorul să ruleze mai multe copii ale aceluiași program, copii denumite instanțe. Fiecare instanță are o valoare *hInstance* diferită).
- *hPrevInstance*

- este identificatorul celei mai recente instanțe anterioare instanței curente. Dacă nu există instanțe anterioare *hPrevInstance* are valoarea 0/NULL

- *szCmdLine*
  – este o variabilă pointer care punctează către un şir de caractere ce conține parametrii din linia de comandă şi care sunt transferați programului
- *iCmdShow*
  – este un număr ce reprezintă modul de afişare al primei ferestre executate (1=normal/7=pictogramă)

(2) Crearea unei ferestre

Definirea clasei de ferestre

O fereastră este creată dintr-o clasă *window*. O astfel de clasă se crează prin definirea caracteristicilor ei, printer care şi a procedurii de fereastră care prelucrează mesajele transmise de fereastra creată.

După crearea clasei este necesar să o înregistrăm prin apelarea funcției **RegisterClass**. Doar prima instanță a unui program trebuie să înregistreze clasa *window*.

WNDCLASS este o structură care îmi permite să definesc clase de obiecte de tip fereastră. Acest tip conține 10 câmpuri (variabile).care descriu caracteristicile tuturor ferestrelor create pe baza acestei clase.

Nume câmp	Semnificație
Style	Stilul ferestrei
lpfnWndProc	Numele procedurii de fereastră.
cbClsExtra cbWndExtra	Două câmpuri care permit rezervarea de spațiu suplimentar în structura *class* şi în structura *window*. Un program poate utiliza acest spațiu în scopuri personale. În general nu avem nevoie de acest spațiu suplimentar şi setăm câmpurile la 0.
hInstance	Identificatorul instanței ferestrei
hIcon	Stabileşte o pictogramă pentru toate ferestrele create pe baza clasei *window*.
hCursor	Stabileşte o pictogramă pentru cursor
hbrBackground	Specifică culoarea de fundal a ferestrei
lpszMenuName	Specifică meniul feresteri
lpszClassName	Specifică numele ferestrei

Pentru crearea unei clase de ferestre trebuie inițializate aceste 10 câmpuri. Acest lucru se face numai la început, înainte de crearea primei ferestre (deci înainte de prima instanță). După inițializarea câmpurilor trebuie înregistrată clasa cu ajutorul funcției **RegisterClass()**. Acest lucru se face de asemenea doar pentru prima instanță. De aceea înainte de aceste operații se verifică condiția **hPrevInstance= =NULL** (adică nu există instanță anterioară).

În continuare este explicată inițializarea câmpurilor în cazul programului minapp.cpp

Stilul ferestrei

```
wcmain.style = CS_BYTEALIGNWINDOW|CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS
```

fereastra va fi reactualizată complet la fiecare modificare orizontală sau verticală şi recunoaşte double-click de mouse (CS = class style)

Procedura de fereastră este **WndProc**. Ea tratează mesajele transmise de fereastra curentă

```
wcmain.lpfnWndProc = WndProc
```

Instanța ferestrei: este instanța curentă **hInstance** (dată de primul parametru al WinMain())

```
wcmain.hInstance = hInstance;
```

Pictograma pentru toate ferestrele create pe baza clasei **wcmain** se obține prin încărcarea unei pictograme (o imagine de tip *bitmap*) cu funcția **LoadIcon**. Pentru obținerea unei pictograme predefinite se foloseşte funcția LoadIcon cu primul parametru NULL, iar al doilea este un identificator cu prefixul IDI (IDentificator de Icon) şi este definit în fişierul **windows.h**

```
wcmain.hIcon = LoadIcon(NULL, IDI_APPLICATION);
```

Cursorul pentru fereastră se obține prin încărcarea unui cursor (o imagine de tip *bitmap*) cu funcția **LoadCursor**. Pentru obținerea unui cursor predefinit valoarea primul parametru al funcției este NULL iar al doilea este un identificator care începe cu prefixul IDC şi este definit în fişierul **windows.h**

```
wcmain.hCursor = LoadCursor(NULL, IDC_ARROW);
```

hbrBackground este un *handle* către o pensulă (**hbr** = **h** + **br** unde **h** este pentru *handle* şi **br** pentru *brush*) pentru culoarea de fundal. O pensulă reprezintă un model colorat de pixeli utilizat în umplerea unei zone. Există mai multe pensule standard, care pot fi obținute cu funcția **GetStockObject** (cu care pot fi obținute şi alte obiecte grafice).

```
wcmain.hbrBackground = (HBRUSH) GetStockObject(WHITE_BRUSH);
```

Menu:

```
wcmain.lpszMenuName = MAKEINTRESOURCE(IDR_DBVAPP);
```

Dacă nu avem menu atunci: `wcmain.lpszMenuName = NULL;`

Numele clasei:- definesc numele clasei de ferstre

```
wcmain.lpszClassName= szKlassenName;
```

Urmează înregistrarea clasei de ferestre, prin transmiterea adresei variabilei **wcmain**:

RegisterClass(&wcmain) şi se testează dacă a putut fi înregistrată clasa. În caz contrar se returnează mesaj de eroare.

Crearea unei ferestre din clasa definită

Crearea unei instanțe a clasei definite anterior se realizează cu ajutorul funcției **CreateWindow**. Această funcție returnează un *handle* (o referință) la obiectul fereastră creat.

Funcția **CreateWindow** are ca parametri:

- ***numele clasei*** de ferestre căreia îi aparține obiectul fereastră creat: `szClassName`
- ***titlul ferestrei*** (care apare în bara de titlu): `szWindowTitle`
- ***stilul ferestrei***: `WS_SYSMENU|WS_CAPTION|WS_OVERLAPPED|WS_BORDER|WS_MINIMIZEBOX,`
- ***coordonatele*** colțului stânga sus: *xPos, yPos*
- ***dimensiunea*** ferestrei: *width, height*
- ***referința către fereastra părinte***: **NULL** (pentru că nu avem fereastră părinte)
- ***referința către meniu***: `LoadMenu(hInstance, MAKEINTRESOURCE(IDR_DBVAPP))`
- ***instanța*** căreia îi aparține fereastra: *hInstance* (parametru al funcției **WinMain**)
- ***creation parameters***: **NULL**

Referința (*handle*) către fereastra creată este în exemplul de mai sus *hwndmain* şi este de tip **HWND** (*handle to a window*). Fiecare fereastră are un *handle* propriu, pe baza căruia se poate stabili de unde vine un mesaj sau unde se efectuează anumite operații. Pot crea mai multe ferestre pe baza aceleiaşi clase, dar fiecare va avea un *handle* diferit.

După ce creez fereastra cu ajutorul funcției **CreateWindow**, testez dacă programul a putut să o creeze. Altfel (`if(hwndmain==NULL)`) se returnează eroare.

(3) Afişarea unei ferestre

După ce am creat obiectul de tip fereastră, acesta trebuie afişat pe ecran. Acest lucru se face prin apelul succesiv a două funcții:

```
ShowWindow(hwndmain, iCmdShow);
```

afişează fereastra pe ecran. Primul parametru specifică fereastra care este afişată, iar al doilea tipul de afişare (`SW_SHOWNORMAL` = normal sau `SW_SHOWMINNOACTIVE` = pictogramă).

```
UpdateWindow(hwndmain);
```

transmite către procedura de fereastră un mesaj de tip `WM_PAINT`, care are ca rezultat redesenarea ferestrei (reactualizarea zonei client a ferestrei).

(4) Bucla de mesaje

Programul trebuie să prelucreze date primite de la utilizator prin intermediul tastaturii şi a mouse-ului. Cu ajutorul tastaturii şi a mouse-ului se produc evenimente de intrare care sunt transmise procedurii de fereastră sub formă de mesaje.

Sistemul **Windows** include o structură de date – *message queue* (coadă de mesaje) – care converteşte fiecare acțiune a utilizatorului într-un mesaj şi îl plasează în coada de mesaje.

Cu ajutorul funcției **GetMessage** pot extrage mesajul aflat la începutul cozii într-o structură de tip **MSG**. Acest mesaj este convertit cu ajutorul funcției **Translate Message** şi apoi acesta este transmis către procedura de fereastră cu ajutorul funcției **DispatchMessage**.

Pentru preluarea mesajelor de fereastră se defineşte o buclă de mesaje, care este activă pe tot parcursul existenței ferestrei:

```
while (GetMessage(&msg, NULL, 0, 0))
{
       TranslateMessage(&msg);   // Traducerea mesajului
       DispatchMessage(&msg);    // Transmiterea mesajului către
                                 //procedura de fereastră
}
```

Variabila *msg* este o structură de tip MSG şi are următoarele câmpuri:

- `HWND hwnd` – identifică fereastra de la care vine mesajul
- `UINT message` – este o constantă ce identifică mesajul. Aceste constante încep cu prefixul WM (Window Message) (de ex. WM_PAINT sau WM_LBUTTONDOWN)
- `WPARAM wParam` – un parametru (pe 16 biți)
- `LPARAM lParam` – un parametru (pe 32 biți). Aceşti 2 param. depind de tipul mesajului şi furnizează informații suplimentare referitoare la mesaj
- `DWORD time` – conține momentul de timp la care mesajul a fost plasat în coada de mesaje
- `POINT pt` – conține coordonatele mouse-ului la momentul generării mesajului

Parametrii 2, 3 şi 4 se setează la NULL respectiv 0, 0 pentru a indica faptul că sunt recepționate mesaje de la toate ferestrele create de program.

Pentru orice mesaj în afară de WM_QUIT funcția **GetMessage**( ) returnează o valoare diferită de 0. Mesajul WM_QUIT determină ieşirea din buclă, iar programul returnează câmpul *wparam* al structurii *msg*.

Prelucrarea mesajelor cu procedura de fereastră

(1) Procedura de fereastră

Procedura de fereastră (în cazul nostru **WndProc**) determină conținutul ce va fi afişată în zona client a ferestrei corespunzătoare şi modul de răspuns al ferestrei la evenimentele produse de utilizator (de ex. click de *mouse*, apăsarea unui buton, selectarea unei opțiuni de meniu etc.). O procedură de fereastră este întotdeauna asociată unei clase *window* particulară şi poate avea orice nume.

Definiția procedurii de fereastră:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
```

Procedura are 4 parametri:

- `HWND hwnd` – identifică fereastra de la care a primit mesajul
- `UINT message` – identifică mesajul primit (de ex. WM_PAINT)
- `WPARAM wParam`
- `LPARAM lParam` – aceşti 2 parametri conțin informații suplimentare despre mesaj şi depind de fiecare mesaj în parte.

(2) Generarea şi prelucrarea mesajelor

Atât sistemul de operare cât şi aplicația în sine pot genera mesaje. care sunt plasate în coada de mesaje ale aplicației (*mesage application queue*). Ulterior aceste mesaje sunt extrase din coadă de către aplicație (prin funcția **GetMessage**( )) şi expediate către procedura de fereastra, care le prelucrează.

Există anumite mesaje care nu sunt plasate în coada de mesaje ci sunt trimise direct către procedura de fereastră (*unqueued messages*). De exemplu funcția **CreateWindow**( ) transmite un mesaj WM_CREATE procedurii de fereastră (fără ca acest mesaj să fie plasat întâi în coada de mesaje).

Extragerea mesajelor din coada de mesaje se face cu ajutorul funcției **GetMessage** în bucla de mesaje discutată mai sus.

Fiecare mesaj este prelucrat de către procedura de fereastră. Acest lucru se întâmplă într-o construcție de tip `switch`, care în funcție de mesaj efectuează anumite operații (stabilite de către programator).

***Exemplu*:** Scrie în fereastră mesajul “Primul program”

```
switch (message)
{
  case WM_PAINT:
      HDC      hdc; //contextul grafic
      PAINTSTRUCT   ps;
      RECT rect; //Obiect dreptunghi

      hdc = BeginPaint(hwnd, &ps); //Obţinerea contextului grafic
      GetClientRect(hwnd,&rect); //Obţinerea suprafeţei de desenare

      //Scrierea unui text în fereastră
      DrawText(hdc,"Primul program",-1,&rect,DT_CENTER|DT_VCENTER);
      EndPaint(hwnd, &ps);

      return 0;

   case WM_DESTROY:
       PostQuitMessage(0);
       return 0;

 default:  // Alte mesaje sunt trimise către funcţia implicită de
          //tratare
          return DefWindowProc(hwnd, message, wParam, lParam);
}
```

În general programatorul va stabili operaţii numai pentru anumite mesaje, caz în care se va returna valoarea 0 (adică mesajul a fost prelucrat).

Mesajele care nu sunt prelucrate în mod explicit de către procedura de fereastră, trebuie pasate către funcţia implicită de prelucrare a mesajelor **DefWindowProc.**

(3) Mesajul WM_PAINT

Acest mesaj generează redesenarea zonei client a ferestre. El este transmis atunci când o anumită porţiune a zonei client este invalidă şi trebuie redesenată, de exemplu atunci când:

- se creează o fereastră pentru prima dată (generat de funcţia **UpdateWindow**)
- se redimensionează fereastra (atunci când câmpul *style* al structurii *wcmain* a fost definit de tip CS_HREDRAW şi CS_VREDRAW – adică invalidarea întregii ferestre în cazul redimensionării orizontale sau verticale)
- se minimizează şi apoi se restaurează fereastra
- atunci când se deplasează fereastra
- atunci când se revine la fereastra aplicaţiei după ce aceasta a fost acoperită de alte ferestre

Obținerea și eliberarea contextului graphic (BeginPaint / EndPaint)

Pentru a putea desena în fereastră este necesară obținerea unui context graphic (device context) păstrat într-o variabilă `hdc` de tip `HDC`. Pentru aceasta se apelează funcția **BeginPaint**:

```
hdc = BeginPaint(hwnd, &ps);
```

Primul parametru *hwnd* identifică fereastra care a generat mesajul.

Al doilea parametru este un *pointer* către o structură de tip PAINTSTRUCT, care conține informații ce pot fi utilizate la redesenarea ferestrei. Câmpurile structurii PAINTSTRUCT sunt:

- HDC hdc – *handle* către un dispozitiv de ieşire (vezi mai jos)
- BOOL fErase – pentru ştergerea zonei client
- RECT rcPaint – dreptunghiul care reprezintă zona invalidă a zonei client. Acest câmp îmi permite de ex. să realizeze redesenarea numai a porțiunii invalide, nu neapărat a întregii zone client.
- BOOL fRestore
- BOOL fIncUpdate
- BYTE rgbReserved[6]

ultimele trei câmpuri sunt rezervate pentru sistemul Windows.

Aceste câmpuri sunt inițializate de către funcția **BeginPaint**.

Funcția **BeginPaint** realizează ştergerea zonei client prin acoperirea ei cu culoarea setată la câmpul **hbrBackground** al structurii *wcmain*. Apoi este validată zona client şi se înapoiază o variabilă *hdc* denumită *handle to a device context*, adică un *handle* către un dispozitiv de ieşire (de ex. ecran sau imprimantă).

Înainte de încheierea prelucrării mesajului, trebuie eliberat contextual graphic prin intermediul funcției **EndPaint**.

```
EndPaint(hwnd,&ps) ;
```

Obținerea zonei client pentru desenare

Funcția **`GetClientRect`**`(hwnd, &rect)` permite plasarea în variabila `rect` (structură de tip *rectangle*) a dimensiunilor zonei client a ferestrei. Structura `rect` conține 4 câmpuri: `left, top, right` şi `bottom`.

`Left` şi `top` sunt fixate la valoarea 0 şi reprezintă colțul din stânga sus al zonei client. `Right` şi `bottom` conțin lățimea şi respectiv înălțimea zonei client (dau colțul dreapta jos) măsurate în pixeli.

Mesajul WM_DESTROY

Mesajul rezultă în urma selectării opțiunii *Close* din meniul **System** aferent programului sau prin apăsarea combinației de taste Alt+F4 sau dacă am o opțiune de meniu/buton căruia îi asociez distrugerea ferestrei.

Acest mesaj semnalează programului că urmează distrugerea ferestrei.

Tratarea mesajului se face în mod standard prin apelarea funcției **PostQuitMessage**(0) care inserează în coada de mesaje un mesaj **WM_QUIT**. Atunci când funcția **GetMessage** va extrage mesajul **WM_QUIT**, va returna valoarea 0, ceea ce va determina încheierea buclei de mesaje şi implicit terminarea programului.

2. Elemente de grafică şi afişare de text

2.1. Despre zona client

Zona client ocupă tot spațiul ferestrei ce nu include bara de titlu, cadrul de redimensionare, bar de meniu şi barele de navigare. Zona client reprezintă acea porțiune de fereastră în care utilizatorul poate efectua diverse operații grafice.

În general operațiile grafice sunt efectuale în urma generării unui mesaj de tip **WM_PAINT** (dar nu numai). Acest mesaj informează procedura de fereastră că a fost invalidată o porțiune a zonei client şi că aceasta este pregătită pentru reactualizare (deci pentru efectuarea de operații grafice).

Un mesaj **WM_PAINT** este generat în următoarele cazuri:

- o zonă a ferestrei acoperită anterior este readusă în prim plan
- fereastra este redimensionată, iar stilul ferestrei este de tip **CS_HREDRAW/CS_VREDRAV**
- sunt utilizate bare de navigare
- programul foloseşte funcția **InvalidateRect**/**InvalidateRegion** pentru a genera în mod explicit un mesaj **WM_PAINT**
- este închisă o casetă de dialog/mesaj care se suprapunea peste zona client
- un meniu popup este selectat şi apoi eliberat
- etc.

Observație: observăm că putem genera în mod explicit un mesaj **WM_PAINT** prin apelul funcției **InvalidateRect**!!!

Dimensiunile zonei client

Dimensiunile zonei client variază de la fereastră la fereastră şi depind evident de dimensiunea ferestrei. Ori de câte ori se modifică dimensiunea ferestrei, sistemul Windows trimite un mesaj **WM_SIZE** procedurii de fereastră.

În acest caz variabila dată de parametrul procedurii de fereastră, *lParam*, conține dimensiunile zonei client, şi anume:

- `LOWORD(lParam)` = lățimea ferestrei
- `HIWORD(lParam)` = înălțimea ferestrei.

Obținerea acestor dimensiuni se poate face cu următorul cod:

```
static short cxClient, cyClient;
//alte instrucțiuni, iar în instrucțiunea
switch(message)
{
      //alte mesaje
```

```
        case WM_SIZE:
              cxClient = LOWORD(lParam);
              cyClient = HIWORD(lParam);
        return 0;

        //alte mesaje
        default:
              return DefWindowProc(hwnd, message, wParam, lParam);
}
```

Observație: variabilele `cxClient` şi `cyClient` conţin dimensiunile zonei client şi sunt de tip `static` pentru a putea fi utilizate ulterior la prelucrarea altor mesaje.

2.2. Contextul dispozitiv (*Display Context – DC*)

La apelul funcției **BeginPaint** se obține o variabilă *hdc* de tip HDC. Aceasta ne permite să efectuăm operații grafice pe un anumit dispozitiv de ieşire, care în cazul nostru este ecranul.

Un *handle* reprezintă un număr pe care îl foloseşte sistemul Windows pentru a se referi la un obiect.

DC reprezintă o structură de date asociat unui dispozitiv de ieşire (ex: imprimantă, placă video- în acest caz este asociat unei ferestre de pe ecran, plotter).

Mediul de operare grafic utilizează valorile din structura *DC* (atributele contextului aferent dispozitivului de ieşire) pentru a determina diferite elemente (cum ar fi: culoare, culoare de fundal, font, pensulă, coordonate etc.)

Când un program vrea să deseneze un obiect, trebuie mai întâi să obțină un *hdc*, iar când a terminat operațiile de desenare va trebui să-l elibereze. Acest lucru se poate realiza în două moduri:

1. în cazul tratării mesajului **WM_PAINT** cu perechea de funcții **BeginPaint**, **EndPaint**
2. în cazul tratării altui mesaj, prin perechea de funcții **GetDC**, **ReleaseDC**:

***Exemplu*:**

```
hdc=GetDC(hwnd);
// funcţii de desenare
ReleaseDC(hwnd,hdc);
```

Observație: În cazul 1. când se tratează mesajul de tip **WM_PAINT** se redesenează doar porțiunea invalidă din zona client, nu întreaga zonă client. Dacă se doreşte redesenarea întregii zone client trebuie apelată funcția

```
InvalidateRect(hwnd, NULL, TRUE)
```

(cu TRUE dacă se doreşte ştergerea fundalului şi cu FALSE dacă nu se doreşte ştergerea fundalului) înainte de apelul funcției **BeginPaint**

În al 2-lea caz se poate desena în toată zona client.

2.3. Afişarea de text

Există două funcții pentru afişarea de text

```
(1) int DrawText(HDC hdc, LPCSTR lpsz, int nc, RECT FAR *lpcr, UINT format)
```

unde

- parametrul 3 (nc) conține nr. de caractere al şirului ce trebuie afişat. În cazul şirurilor de caractere care în C se termină cu '\0', pentru ca întreg şirul să fie afişat setez nc=-1.
- parametrul 4 conține adresa unui dreptunghi de formatare (în funcție de acesta pot de exemplu alinia textul la dreapta, stânga, centrat etc.)
- parametrul 5: un şir format o serie de indicatori (legați prin | = „sau" pe biți) de format

de exemplu:

DT_TOP, DT_BOTTOM, DT_LEFT, DT_RIGHT	- aliniere sus-jos, stânga-dreapta
DT_CENTER, DT_VCENTER	- centrare orizontală şi verticală
DT_SINGLELINE	- text pe o singură linie

etc.

***Exemplu*:**

```
case WM_PAINT:
        HDC      hdc; //contextul grafic
        PAINTSTRUCT   ps;
        RECT rect; //Obiect dreptunghi

        hdc = BeginPaint(hwnd, &ps); //Obținerea contextului grafic
        GetClientRect(hwnd,&rect); //Obținerea suprafeței de desenare

        //Scrierea unui text în fereastră
        DrawText(hdc,"Primul program",-1,&rect,DT_CENTER|DT_VCENTER);
        EndPaint(hwnd, &ps);

        return 0;
```

```
(2) BOOL TextOut(HDC hdc, int nXStart, int nYStart, LPCSTR lpszString, int
                                                                    cbString)
```

unde:

- `hdc` – handle către contextul de afişare
- `nXStart, nYStart` – coordonatele de la care se începe afişarea

- `lpszString` – şirul de caractere care se afişează
- `cbString` – nr. de caractere care se afişează (lungimea şirului)

Exemplu:

```
case WM_PAINT:
         HDC      hdc;
         PAINTSTRUCT   ps;
         RECT rect;
         char sir[40];
         short lungime=sprintf(sir, „Valorile obtinute sunt %d, %d si
                        %d“,x1,x2,x3); //unde x1, x3, x3 sunt niste
                                       //variablie din program

         hdc = BeginPaint(hwnd, &ps); //Obţinerea contextului grafic
         GetClientRect(hwnd,&rect); //Obţinerea suprafeţei de desenare
         //Scrierea unui text în fereastră


         TextOut(hdc,50,50,sir,lungime);
         EndPaint(hwnd, &ps);

         return 0;
```

Şiruri de caractere

Majoritatea funcţiilor WinApi care permit afişarea de text, utilizează tipul LPCSTR. Pot transforma un sir `char* s=”sir de caractere”` la tipul acesta în două moduri:

- `TEXT(„sir de caractere”)`
- `L”sir de caractere”`

Dacă vreau sa afişez un şir de caractere care conţine şi valorile unor variabile pot folosi afişarea cu format, în care utilizez specificatori de format, similar ca în C. Pentru aceasta utilizez clasa `CString` pentru care trebuie importate

```
#include <atlbase.h>
#include <atlstr.h>
```

***Exemplu*:**

```
CString sir;
sir.format(„Avem variabila a=%d si b=%lf”,a,b);
DrawText(hdc,sir,-1,&rect,DT_CENTER|DT_VCENTER);
```

2.4. Operații grafice

(1) Culori:

O culoare este un obiect de tip COLORREF.

Fiecare culoare este alcătuită din trei culori de bază, care sunt amestecate în diferite proporţii. Cele trei culori de bază sunt roşu (R = *red*), verde (G = *green*) şi albastru (B = *blue*). Acest sistem se numeşte sistemul RGB.

O culoare se obține prin funcția `RGB(r,g,b)` unde `r` = valoarea pentru roşu, `g` = valoarea pentru verde, `b` = valoarea pentru albastru.

Fiecare variabilă de culoare poate avea valori între 0 şi 255, acestea reprezentând „cantitatea de culoare" din culoarea respectivă (nuanța). Valoarea 0 apropie culoarea de negru, iar 255 reprezintă culoarea „pură".

Dacă am r = g = b obțin diferite tonuri de gri, unde (0,0,0) este negru şi (255,255,255) este alb.

Dacă două dintre componente sunt egale, de exemplu g = b, obțin nuanțe din componenta a treia, în cazul nostru de roşu. Cu cât valorile g = b sunt mai mari (mai aproape de 255) obțin o nuanță mai deschisă de roz, mai aproape de alb.

(2) Peniţe şi pensule

Se face cu „penița" (pen) curent iar umplerea de suprafețe se face cu „pensula" (brush) curentă. Alegerea acestora se va discuta în cele ce urmează:

(a) C***rearea unei peniţe***:

```
HPEN CreatePen(int fnPenStyle, int nWidth, COLORREF color);
```

unde `fnPenStyle` – stilul care poate fi:

- PS_SOLID : linie continuă
- PS_DASH : linie întreruptă
- PS_DOT : linie punctată
- PS_DASHDOT: linie formată din puncte şi liniuțe
- etc.

`nWidth` – lățimea liniei (în pixeli)

`color` – culoarea folosită (de tip RGB)

(b) ***Crearea unei pensule*** se poate face cu 4 funcții diferite, dintre care vom prezenta două:

Pensulă cu umplere uniformă:

```
HBRUSH CreateSolidBrush(COLORREF color);
```

Pensulă haşurată:

```
HBRUSH CreateHatchBrush(int fnBrushStyle, COLORREF color);
```

unde `fnBrushStyle` – stilul pensulei poate fi:

- HS_HORIZONTAL
- HS_VERTICAL
- HS_BDIAGONAL
- HS_FDIAGONAL
- HS_CROSS
- HS_DIAGCROSS

Aceste funcții creează obiectele respective (pen sau brush). Pentru a utiliza efectiv un astfel de obiect el trebuie selectat ca fiind obiectul curent cu ajutorul funcției `SelectObject`.

Atunci când un obiect `HPEN` sau `HBRUSH` este selectat, el devine obiectul curent, desenarea efectuându-se cu el.

Atenție: Funcția `SelectObject` returnează obiectul (pen /brush/alt obiect) curent anterior, pentru ca acesta să poată fi selectat din nou, după ce nu mai este necesar obiectul creat de noi. Atunci când nu mai avem nevoie de un obiect creat de noi, se recomandă restaurarea obiectului anterior şi distrugerea obiectului nostru.

***Exemplu*:**

//în cadrul procedurii de fereastră:

```
HPEN pen;
HPEN old_pen;
HBRUSH br1, br2, old_br;

switch (message)
{
      case WM_PAINT:
            hdc = BeginPaint(hwnd,&ps);
            GetClientRect(hwnd,&rect);

            //crearea noii penite
            pen = CreatePen(PS_SOLID,2,RGB(0,0,255));

             //pastrarea penitei vechi
            old_pen=(HPEN)SelectObject(hdc,pen);

            //crearea a doua pensule noi
            br1=CreateSolidBrush(RGB(255,33,33));
            br2=CreateHatchBrush(HS_CROSS, RGB(33,255,33));

            //pastrarea pensulei vechi
            old_br=(HBRUSH)SelectObject(hdc,br1);
```

```
            //instructiuni de desenare
            //…
            //restaurarea obiectelor anterioare
            SelectObject(hdc,old_pen);
            SelectObject(hdc,old_br);

            //stergerea obiectelor create
            DeleteObject(pen);
            DeleteObject(br1);
            DeleteObject(br2);
            EndPaint(hwnd,&ps);
            return 0;
      //celelalte optiuni din switch
      default:
            return DefWindowProc(hwnd,message,wParam,lParam);
}
```

(3) Desenarea de puncte (pixeli):

```
SetPixel(HDC hdc, int x, int y, COLORREF color);
```

Exemplu:

```
      case WM_PAINT:
            hdc=BeginPaint(hwnd,&ps);
            GetClientRect(hwnd,&rect);

            //Desenarea unei sinusoide
            coef=(rect.bottom-rect.top)/2;
            for(x=rect.left;x<=rect.right;x++)
             {
                  y=coef*(sin(3.14*x/180)+1);
                  SetPixel(hdc,x,y,RGB(0,0,0));
             }
            EndPaint(hwnd,&ps);
            return 0;
```

(4) Desenarea liniilor:

```
MoveToEx(HDC hdc, int x, int y, LPPOINT pt);

LineTo(HDC hdc, int x, int y);
```

Funcția `MoveToEx()` permite poziționarea într-un anumit punct al ferestrei, dat prin coordonatele `(x,y)`. Acest punct va fi punctul de start pentru linia care se va trasa.

Funcția `LineTo()` permite trasarea unei linii de la punctul curent (la care ne-am deplasat de exemplu cu funcția `MoveToEx()` sau cu altă instrucțiune de desenare) la punctul determinat de coordonatele `(x,y)` date de parametrii funcției.

Observație: Ultimului parametru în cazul funcției **MoveToEx** i se va da valoarea NULL.

Exemplu:

```
case WM_PAINT:
      hdc=BeginPaint(hwnd,&ps);
      GetClientRect(hwnd,&rect);

      //Desenarea unei sinusoide
      coef=(rect.bottom-rect.top)/2;

      MoveToEx(hdc,0,coef,NULL);
      for(x=rect.left+1;x<=rect.right;x++)
       {
            y=coef*(sin(3.14*x/180)+1);
            LineTo(hdc,x,y);
       }
      EndPaint(hwnd,&ps);
      return 0;
```

(5) Desenarea dreptunghiurilor:

```
Rectangle(HDC hdc, int x1, int y1, int x2, int y2);
```

unde: $(x_1, y_1), (x_2, y_2)$ definesc coordonatele colțurilor stânga-sus şi dreapta-jos ale dreptunghiul.

(6) Desenarea elipselor (cercurilor):

```
Ellipse(HDC hdc, int x1, int y1, int x2, int y2);
```

unde: $(x_1, y_1), (x_2, y_2)$ definesc dreptunghiul circumscris elipsei. (x_1, y_1) sunt coordonatele colțului stânga sus (sau dreapta jos) al dreptunghiului circumscris, iar (x_2, y_2) sunt coordonatele colțului diagonal opus lui (x_1, y_1).

Pentru a desena un cerc este suficient ca: $|x_1-x_2|=|y_1-y_2|$, adică dreptunghiul circumscris să fie pătrat.

Observație: contururile formelor vor fi desenate cu penița curentă iar umplerea se va face cu pensula curentă. În mod predefinit penița curentă este linie continuă, neagră de grosime 1 iar pensula curentă are culoarea fundalului (în general alb);

3. Operații cu Mouse-ul

Vom considera *mouse*-uri cu 2 butoane (butonul stâng şi butonul drept).

Există următoarele acțiuni care pot avea loc în cazul unui *mouse*:

- *click*: - apăsare şi eliberare a unui buton de *mouse*
- *double-click*: - apăsare şi eliberare de două ori în interval foarte scurt de timp a unui buton de *mouse*
- *move*: - deplasarea *mouse*-ului
- *drag*: - deplasarea *mouse*-ului în timp ce se ține apăsată una dintre taste

Pentru butoane se utilizează următoarele prescurtări:

- **LBUTTON** – butonul stâng
- **RBUTTON** - butonul drept.

În cazul acțiunii *mouse*-ului, fereastra noastră primeşte mesaje corespunzătoare, în funcție de acțiune şi de locația la care se afla *mouse*-ul. Sunt prezentate în continuare numai acelea dintre mesajele generate de *mouse* care pot prezenta interes.

Procedura de fereastră primeşte mesaje de la *mouse* şi atunci când *mouse*-ul este poziționat în zona client a ferestrei, dar şi atunci când este poziționat în afara zonei client. Sunt prezentate doar mesajele primite atunci când ne aflăm în zona client.

Listă de mesaje:

Mesaj	Descriere
WM_LBUTTONDOWN	butonul stâng este apăsat
WM_RBUTTONDOWN	butonul drept este apăsat
WM_LBUTTONUP	butonul stâng este eliberat
WM_RBUTTONUP	butonul drept este eliberat
WM_LBUTTONDBLCLK	a doua apăsare a butonului stâng
WM_RBUTTONDBLCLK	a doua apăsare a butonului drept
WM_MOUSEMOVE	mouse-ul este deplasat

Atunci când procedura de fereastră primeşte un mesaj de la *mouse*, parametrul `lParam` conține coordonatele poziției *mouse*-ului la momentul generării mesajului. Parametrul

`lParam` este împărțit în două secvențe a câte 16 biți: partea *low* (inferioară) – conține coordonata orizontală x - şi parte *high* (superioară) – conține coordonata verticală y. Acestea se obțin cu ajutorul macrourilor `LOWORD()` şi `HIGHWORD()`.

```
x = LOWORD(lParam);
y = HIWORD(lParam);
```

Exemplu:

Scriem doar procedura de fereastră, restul este similar ca la primul exemplu:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM
                                                                  lParam)
{
      POINT p1;
      static POINT p2;
      HDC hdc;
      RECT rect;

   switch (message)
   {
       case WM_LBUTTONDOWN:
            p1.x=LOWORD(lParam);
            p1.y=HIWORD(lParam);

            hdc=GetDC(hwnd); //Obtinerea contextului grafic

            Ellipse(hdc,p2.x-5,p2.y-5,p2.x+5,p2.y+5);

            MoveToEx(hdc,p2.x,p2.y,NULL);
            LineTo(hdc,p1.x,p1.y);

            p2=p1;
            ReleaseDC(hwnd,hdc); //Eliberarea contextului grafic
            return 0;

       case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

      default:  return DefWindowProc(hwnd, message, wParam, lParam);
   }

  return 0;
}
```

Parametrul *wParam* conține informații referitoare la apăsarea anumitor taste (SHIFT sau CTRL) şi la apăsarea butoanelor de *mouse*. Acest parametru poate fi o combinație (sau pe biți) între valorile:

MK_CONTROL	- tasta control este apăsată
MK_SHIFT	- tasta shift este apăsată

MK_LBUTTON - butonul stâng al mouse-ului este apăsat

MK_RBUTTON - butonul drept al mouse-ului este apăsat.

Obs:

- dacă `(MK_CONTROL & wParam)` are valoarea *true*, înseamnă că tasta CTRL este apăsată

- dacă `(MK_SHIFT & wParam)` are valoarea *true*, înseamnă că tasta SHIFT este apăsată

etc.

Deci pentru a stabili dacă una dintre taste este apăsată (sau mai multe) este suficient să testăm cu un **`if( )`** valoarea „şi”-ului pe biți dintre parametrul *wParam* şi valoarea care ne interesează.

Exemplu:

```
static POINT p1, p2;

switch (message)
   {
      case WM_LBUTTONDOWN:
            p1.x=LOWORD(lParam);
            p1.y=HIWORD(lParam);
            return 0;

      case WM_MOUSEMOVE:
             if(wParam & MK_LBUTTON)
             {
                   hdc=GetDC(hwnd);
                   p2.x=LOWORD(lParam);
                   p2.y=HIWORD(lParam);

                   if (wParam & MK_SHIFT)
                        Ellipse(hdc,p2.x,p2.y,p3.x,p3.y);
                   else
                        Rectangle(hdc,p2.x,p2.y,p3.x,p3.y);

                   ReleaseDC(hwnd,hdc);
             }
             return 0;
      default: return DefWindowProc(hwnd, message, wParam, lParam);

}
```

Observații:

1. **Privind mesajul WM_MOUSEMOVE**:

   Atunci când *mouse*-ul este deplasat se generează la un interval foarte mic o multitudine de mesaje **WM_MOUSEMOVE**. Sistemul nu le poate prelucra cu aceeaşi viteză cu care sunt generate. Pentru a evita supraîncărcarea cozii de mesaje, atunci când la primirea unui mesaj **WM_MOUSEMOVE** în coada de mesaje se mai află un

astfel de mesaj, acesta dinainte este şters. Astfel încât numărul de mesaje de acest tip care sunt prelucrate efectiv este mai mic decât numărul de mesaje generate în total la deplasarea cursorului.

2. **Privind double-click**:

   Un mesaj **WM_LBUTTONDBLCLK** (respectiv **WM_RBUTTONDBLCLK**) va fi generat numai dacă durata dintre două apăsări succesive a butonului este suficient de mică şi dacă în clasa ferestrei stilul a fost definit ca **CS_DBLCLKS**:

Exemplu:

```
wcmain.style=CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
```

În acest caz apăsarea de două ori la interval scurt a butonului stâng va genera următoarea secvență de mesaje:

WM_LBUTTONDOWN	- la prima apăsare a butonului
WM_LBUTTONUP	- la eliberarea butonului
WM_LBUTTONDBLCLK	- la a doua apăsare a butonului
WM_LBUTTONUP	- la eliberarea butonului

Structura POINT

Structura POINT reprezintă un obiect de tip punct, cu două coordonate, x şi y. Structura are deci două câmpuri x şi y şi este definită în fişierul windows.h.

5. Casete de dialog

5.1. Crearea unei casete de dialog

Se alege opțiunea de menu *Insert-Resource-Dialog* (sau în *Resource-View* cu click-dreapta). Apare un *Wizzard* pentru crearea unui dialog. Cu ajutorul acestuia se pot adăuga diferite controale, precum butoane, casete de validare, butoane radio, casete text etc.

Fiecare control are un număr de identificare (ID).

De asemenea caseta de dialog primeste un ID.

5.2. Apelarea unei casete de dialog se realizează cu ajutorul funcției

```
BOOL CALLBACK DialogBoxParam(HI NSTANCE hInstance, LPCSTR nume_box,
       HWND parent, DLGPROC ProcDialog, LPARAM dwInitParam);
```

sau

```
BOOL CALLBACK DialogBoxParam(HI NSTANCE hInstance, LPCSTR nume_box,
                 HWND parent, DLGPROC ProcDialog);
```

unde:

- `hInstance` – instanța programului
- `nume_box` – numele casetei de dialog, care în general se obține cu `MAKEINTRESOURCE(IDD_DIALOG)` (`IDD_DIALOG` ID-ul casetei de dialog)
- `parent` – fereastra de care aparține caseta de dialog
- `ProcDialog` – procedura de ferastră asociată casetei de dialog
- `dwInitParam` – primeşte valoarea NULL

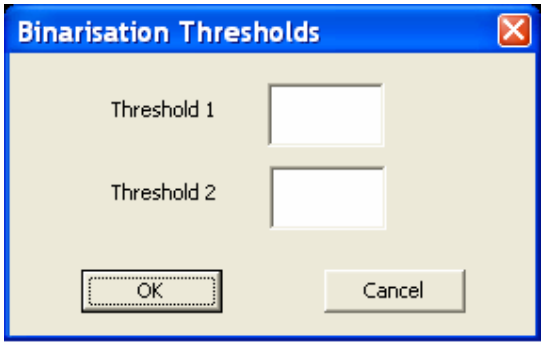
Funcția returnează *true* sau *false*, în funcție de al doilea parametru al funcției `EndDialog()`, care închide dialogul. Acest fapt permite de exemplu returnarea valorii true când se apasă butonul OK (dacă există) şi a valori false când se apasă butonul CANCEL (dacă există).

5.3. Procedura de ferastră

Se aseamănă cu procedura de ferastră pentru fereastra principală. În mesajul WM_COMMAND pot fi tratate mesajele generate de către diferitele controale din caseta de dialog.

Exemplu: Binarizare. Se citesc valorile pragurilot T1 şi T2 cu ajutorul unei casete de dialog, iar apoi se utilizează în cadrul funcției de binarizare (care NU este prezentată în exemplu).

Caseta de dialog arată astfel:



Fişierul care conţine procedura de fereastră pentru caseta de dialog şi funcţia BinaryImage

```
#include <windows.h>
#include "definitions.h"
#include "resource1.h"

int T1,T2;  //Cele două praguri citite cu ajutorul casetei de dialog

extern HINSTANCE myinstance; //Instanta programului declarată global în
                             //programul principal
extern HWND hwndmain; //handle-ul către fereastra principală
//extern MyGImage prelImage;

BYTE *BinaryImage(BYTE *buffer)
{
      //Apelul casetei de dialog cu ID-ul IDD_BINAR

      if(DialogBoxParam(myinstance, MAKEINTRESOURCE(IDD_BINAR), hwndmain,
            (DLGPROC)WndBINARDialogProc, NULL)==TRUE)

            if(T2>T1)
                  return binary(image,T1,T2);
            else
                   return binary(image,T2,T1);
      else return NULL; //Dacă s-a apăsat butonul Cancel

}

... ... ... ...

//Procedura de fereastră a casetei de dialog
BOOL CALLBACK WndBINARDialogProc(HWND hdlg, UINT message, WPARAM wParam,
LPARAM lParam)
{

      int bState;
      static int ok_cancel=TRUE; //stabileste dacă s-a închis cu OK sau
                                 //Cancel
      switch (message)
      {

            case WM_DESTROY:
                 EndDialog(hdlg,ok_cancel);
                 return TRUE;
            case WM_COMMAND:
```

```
                switch (LOWORD(wParam))
                {
                      case IDOK: //S-a apăsat OK

                        //Functie care citeste un întreg dintr-o casetă
                        //de text
                        T1 =(int)GetDlgItemInt(hdlg,IDC_T1,&bState,true);
                        if(T1>255) T1=255;
                        if(T1<0) T1=0;
                        T2 =(int)GetDlgItemInt(hdlg,IDC_T2,&bState,true);
                        if(T2>255) T2=255;
                        if(T2<0) T2=0;
                        ok_cancel=TRUE;

                         //Functia care duce la încheierea dialogului
                        EndDialog(hdlg, true);
                        break;

                      case IDCANCEL: //S-a apăsat Cancel
                        ok_cancel=FALSE;
                        EndDialog(hdlg, false);
                        break;
                }

                return TRUE;

      }

      return FALSE;
}
```

5.4. Controale

La caseta de dialog pot fi adăugate diferite controale:

(1) **Butoane**: Când este apăsat un buton se generază un mesaj de tip WM_COMMAND, iar parametrul wParam transmis către procedura de fereastră a dialogului are ca valoatre ID-ul butonului.

(2) **Butoane radio**: Când se selectează un buton radio se generază un mesaj WM_COMMAND, iar parametrul wParam transmis către procedura de fereastră a dialogului are ca valoatre ID-ul

Pentru a creea un grup de butoane radio care se exclud reciproc, trebuie mai întâi creată o group-box, iar apoi se introduc pe rând butoanele radio în aceasta.

(3) **ExitBox**: control de editare

Pentru citirea de valori întregi într-un control de editatare pot fi utilizate următoarele funcții:

```
(int)GetDlgItemInt(hDlg,IDC_Text,NULL,true);
(int)GetDlgItemInt(hDlg,IDC_Text,&nr,true);
```

Ultimul parametru este *true* atunci când se doreşte şi citirea de numere negative. Altfel este *false*.

Pentru citirea unui text (şir de caractere) utilizăm funcția:

```
GetWindowText(GetDlgItem(hDlg,IDC_ControlText),sir_caractere,lu
                        ngime_sir);
```

```
GetDlgItemText()?
```

Handle-ul unui control se obține cu ajutorul funcției:

```
HWND GetDlgItem(HWND hDlg, LPCSTR ID_Control);
```