Curs 9. Visual Basic for Applications (VBA)(2)
1. Matrice
2. Lucrul cu biblioteci DDL şi cu funcţii Windows API

1.Matrici

O matrice poate fi privită ca o mulţime de variabilele de acelaşi tip, având acelaşi nume şi stochează informaţii similare. Fiecărei variabile ce compune matricea îi corespunde un indice (un număr de ordine). Dacă numărul de elemente ale matricei a fost specificat la declarare, matricea se numeşte *statică*, altfel, ea se numeşte *dinamică*.

1.1. Matrici statice

După ce am fixat numărul de elemente la declararea matricei, acesta nu mai poate fi schimbat pe parcurs.
Matricile statice se declară astfel:

```
Dim nume_matrice (ip To iu) As tip_de_date
```

unde:

- nume_matrice – identificatorul (numele) matricei
- ip – indicele primului element
- iu – indicele ultimului element
- tip_de_date – tipul datelor stocate în matrice.

Notă: A nu se confunda matricile statice cu variabilele statice. O variabilă statică e o variabilă declarată cu ajutorul cuvântului cheie *Static* şi îşi păstrează valoarea şi după terminarea procedurii în care a fost declarată, iar o matrice statică este o matrice al cărei număr de elemente a fost specificat la declarare.
La rândul lor, matricile statice pot fi declarate, asemeni variabilelor, cu ajutorul instrucţiunilor *Dim, Static, Private* sau *Public*.

Exemplu: Subrutină ce ilustrează folosirea matricilor statice:

```
Sub MatriceStat ()
Dim I As Integer
Dim iMatrice(1 To 5) As Integer
iMatrice(1) = InputBox("Introduceţi primul element", "Matrice")
For I = 2 To 5
iMatrice (I) = InputBox ("Introduceţi a1 " & I & "-lea element", "Matrice")
Next I
Debug.Print "Primul element este " & iMatrice(1)
For I = 2 To 5
Debug.Print "A1 " & I & "-lea element este " & iMatrice(I)
Next I
End Sub
```

Observaţie. Am folosit funcţia predefinită *InputBox,* care afişează o cutie de dialog ce permite utilizatorului să introducă o valoare, pe care funcţia o returnează. Astfel, după ce fiecare element al matricei a primit câte o valoare introdusă de utilizator, matricea este parcursă cu ajutorul instrucţiunii iterative *For…Next*, pentru a afişa valorile stocate.

Indici

Nu este obligatoriu ca indicele primului element al unei matrici să fie 1; el poate fi orice alt număr întreg.
Exemplu: declaraţia unei matrice de 10 elemente numere întregi se face `Dim iMatrice(5 To 14) As Integer`
Dacă nu specificăm indicele primului element al unei matrici, Access va considera în mod implicit că acesta este zero. Astfel, următoarele două declaraţii sunt echivalente:

```
Dim iMatrice (9) As Integer şi
Dim iMatrice (0 To 9) As Integer
```

Dacă vom dori ca indicele implicit al primului element să nu fie 0, ci altul (1, de exemplu), introducem la secţiunea (Declarations) a modulului următoarea linie de cod:

```
Option Base 1
```

1.2. Matrici dinamice

Dacă nu cunoaştem de la început numărul de elemente pe care trebuie să le aibă o matrice, putem să o declarăm ca fiind dinamică. Astfel de matrice se declară tot cu ajutorul instrucţiunilor *Dim, Static, Private* sau *Public*, dar punând două paranteze rotunde după numele matricei. Apoi, cu ajutorul instrucţiunii *ReDim*, putem fixa indicii primului şi ultimului element al matricei.

Exemplu:

```
Sub MatriceDin()
Dim I As Integer
Dim iTot As Integer
Dim iMatrice() As Integer
iTot = InputBox ( "Introduceţi numărul total de elemente")
ReDim iMatrice(1 To iTot)
iMatrice (1) = InputBox ("introduceţi primul element", "Matrice")
For i = 2 To iTot
iMatrice(i) = InputBox("Introduceţi al "& I & "-lea element", "Matrice")
Next i
Debug.Print "Primul element este " & iMatrice(1)
For i = 2 To iTot
Debug.Print "Al" & I & "-lea element este " & iMatrice(i)
Next i
End Sub
```

După ce am specificat, cu ajutorul instrucţiunii *ReDim*, indicii primului şi ultimului element al matricei, putem folosi din nou instrucţiunea *ReDim* pentru a modifica aceşti indici (şi deci, dimensiunea totală a matricei). In mod normal, modificând aceşti indecşi se vor pierde valorile stocate de elementele matricei. Pentru a preveni acest lucru, folosim cuvântul cheie *Preserve*. Astfel, dacă declarăm următoarea matrice:

```
Dim iMatrice ( ) As Integer
…
ReDim iMatrice ( 1 To 10)
```

şi dorim ulterior să reducem la 5 nr de elemente fără a pierde valorile primelor 5, vom scrie

```
ReDim Preserve iMatrice ( 1 To 5)
```

1.3. Detectarea matricilor

Există trei funcţii care ne ajută să aflăm dacă o variabilă este simplă sau este o matrice: *IsArray, VarType* şi *TypeName.*
Funcţia *IsArray* returnează valoarea *True* dacă variabila care îi este dată drept argument este o matrice şi *False* dacă nu.
Funcţia *VarType* apare şi într-un material anterior, unde am dat şi tabelul cu valorile pe care acesta le poate returna. Din acel tabel reiese că unei variabile de tip matrice îi corespunde valoarea de retur 8192 (sau constanta intrinsecă vbArray). Astfel, dacă variabila dată ca argument funcţiei *VarType* este o matrice, funcţia va avea ca valoare de retur 8192 plus valoarea corespunzătoare tipului variabilelor ce compun matricea.
Exemplu:

```
Sub TestMatrice ( )
Dim iNum As Integer
Dim iMatrice (1 To 10) As Integer
Debug.Print "iNum: " & VarType(iNum)
Debug.Print "iMatrice: " & VarType (iMatrice)
End Sub
```

Dacă vom rula subrutina TestMatrice, în fereastra *Immediate* vor apărea valorile 2 şi 8194. Observăm că pentru variabila *iMatrice*, funcţia *VarType* a returnat valoarea 8194, adică 8192 pentru matrice, plus 2 pentru tipul *Integer* (care e tipul elementelor matricei).
Funcţia *TypeName* se aseamănă cu *VarType*, cu diferenţa că în locul valorii corespunzătoare unui tip de date, ea returnează chiar numele tipului respectiv.
Exemplu:

```
Sub TestMatrice1 ( )
Dim iNum As Integer
Dim iMatrice (1 To 10) As Integer
Debug.Print "iNum: " & TypeName (iMatrice)
Debug.Print "iMatrice: " & TypeName(iMatrice)
```

```
End Sub
```

1.4. Matrici multidimensionale

Până acum am tratat numai matrici unidimensionale, numite şi vectori. Pentru a declara o matrice multidimensională, trebuie să specificăm marginile inferioare şi superioare ale indicilor pentru fiecare dimensiune, separate prin virgulă.
Exemplu: Pentru a declara o matrice bidimensională cu 6 elemente şi cu indicii începând de la 1, scriem:

```
Dim iMatrice (1 To 2, 1 To 3) As Integer
```

Pentru o matrice de aceeaşi dimensiune dar cu indicii începând de la 0 (pp. că nu s-a specificat opţiunea *Option Base 1*):

```
Dim iMatrice (1, 2) As Integer
```

Notă: Teoretic, numărul maxim de dimensiuni ale unei matrici este 60. Practic însă, nu vom folosi matrice cu mai mult de patru dimensiuni, deoarece ar fi foarte greu de lucrat la ele.

1.5. Matrici multidimensionale dinamice

Ca şi cele unidimensionale, matricile multidimensionale pot fi declarate ca fiind dinamice, pentru ca numărul dimensiunilor şi marginilor fiecărei dimensiuni să poată fi stabilite ulterior. Sintaxa este următoarea:

```
Dim iMatrice ( ) As Integer
…
ReDim iMatrice (1 To 2, 1 To 3)
```

Sau, pentru ca indicii matricei să înceapă de la 0:

```
Dim iMatrice ( ) As Integer
…
ReDim iMatrice(1, 2)
```

O astfel de matrice are 2x3 =6 elemente. Similar, prin instrucţiunea următoare: `ReDim iMatrice (1, 2, 3)`

matricea va fi tridimensională şi va conţine 2x3x4 = 24 elemente.

1.6. Accesarea componentelor unei matrici multidimensionale

Pentru a accesa un element al unei matrice multidimensională, trebuie să specificăm indicii corespunzători fiecărei dimensiuni a matricei, ce caracterizează elementul respectiv.
Exemplu: Următoarea subrutină declară o matrice bidimensională şi o umple cu anumite valori, pe care apoi le afişează. Fiecărui element al matricii îi vom da ca valoare suma indicilor care îi corespund.

```
Sub Matrice ( )
Dim I As Integer
Dim j As Integer
‘ crearea matricei dinamice
Dim iMat ( ) As Integer
‘ redimensionarea matricei
ReDim iMat(1 To 2, 1 To 3)
For i = 1 to 2
      For j = 1 to 3
            IMat(i,j)=i+j
      Next
Next
For i = 1 to 2
      For j = 1 to 3
            Debug.Print I & “ + “ & j & “= “ & iMat(i,j)
      Next
Next
End Sub
```

1.7. Stergerea matricilor dinamice

O matrice ocupă în memorie 20 de octeţi, plus câte 4 octeţi pentru fiecare dimensiune, plus numărul de octeţi necesari datelor stocate, adică numărul de elemente ale matricei înmulţit cu numărul de octeţi ocupaţi de tipul respectiv de date. Astfel, matricea iMatrice (1, 2, 3) ocupă în memorie:
20 octeţi _ 3 x 4 octeţi +(2x3x4)x2 octeţi = 80 octeţi.

Pentru a elibera memoria ocupată de o matrice, putem folosi următoarea instrucţiune *Erase* astfel: `Erase iMatrice`

Folosită pentru matrice statice, instrucţiunea *Erase* le va reiniţializa, fără să elibereze memoria ocupată de aceasta.

1.8. Folosirea matricilor ca parametri pentru proceduri

Access permite să dăm şi matrice ca parametru funcţiilor şi subrutinelor. Un parametru-matrice (sau matrice de parametru) dă posibilitatea de a crea proceduri cu un număr variabil de parametri.
Exemplu : următoarea funcţie calculează media aritmetică a numerelor ce îi sunt date ca parametri:

```
Function Media (ParamArray aNum()) As Double
Dim valCrt
Dim db1Suma As Double
For Each valCrt In aNum
      Db1Suma = db1Suma + valCrt
Next
Media = db1Suma / (UBound (aNum) + 1)
End Function
```

Dacă în fereastra *Immediate* scriem:

`?Media (1,2,3,4,5)` şi apăsăm tasta Enter, vom obţine rezultatul 3.

Trei lucruri esenţiale trebuie să le aveţi în vedere când lucrăm cu parametri de tip matrice:

- Pentru a declara o matrice ca parametru al unei proceduri, o precedăm cu cuvântul cheie *ParamArray*.
- Un parametru de tip matrice apărea doar pe ultima poziţie în lista de parametri ai unei proceduri.
- Matricele de parametri pot fi numai de tipul *Variant.*

Funcţia Media calculează întâi suma tuturor elementelor matricei de parametru, pe care o împarte apoi la numărul total de elemente.
Notă: Indicii matricelor de parametrii încep întotdeauna de la zero, indiferent dacă în modulul ce conţine procedura am specificat sau nu opţiunea *Option Base 1*.

Funcţia predefinită UBound returnează valoarea celui mai mare indice al unei matrice. Cum indicii matricei de parametru aNum de la 0, numărul total de elemente este UBound(aNum) + 1.

2. Lucrul cu biblioteci DDL şi cu funcţii Windows API

Prezentăm una dintre cele mai importante facilităţi oferite de VBA: posibilitatea de a apela un DDL (*Dynamic Link Library*) dintr-o procedură VBA. Astfel, putem realiza mai mult decât ne permit funcţiile predefinite şi instrucţiunile VBA: putem controla sistemul de operare Windows.
Windows vine cu mai multe biblioteci DDL, ce conţin sute de funcţii utile pentru programatori. Aceste funcţii formează ceea ce poartă numele de Windows API (*Application Programming Interface*)

Un DLL este un fişier sursă ce conţine un număr mare de funcţii (o bibliotecă de funcţii). Programatorul poate folosi o funcţie dintr-un DLL într-un program de-al său fără ca acel program să conţină o copie a funcţiei respective. Indiferent de numărul aplicaţiilor ce apelează funcţii dintr-un DLL la un moment dat, biblioteca DLL se încarcă în memorie o singură dată, şi anume, prima oară când s-a apelat o funcţie din ea. Apoi, după ce aplicaţiile nu mai au nevoie de funcţiile sale, biblioteca DLL este descărcată din memorie.

2.1. Declararea funcţiilor API

Inainte de a putea apela o funcţie dintr-un DLL, trebuie să-i indicăm lui Access unde se află funcţia şi cum anume să o apeleze. Acest lucru îl realizăm cu ajutorul instrucţiunii *Declare,* incluzând practic o declarare a funcţiei la secţiunea *(General) (Declarations)* a modulului în care se face apelarea. Printr-o astfel de declaraţie, VBA primeşte şase informaţii:

- Domeniul de vizibilitate al funcţiei;
- Numele pe care îl veţi folosi în codul dumneavoastră pentru a apela funcţia;
- Numele şi calea bibliotecii DLL care o conţine;
- Numele funcţiei, aşa cum este ea definită în DLL;
- Numele şi tipul argumentelor;
- Tipul valorii returnate (dacă aceasta există).

Dacă funcţia returnează o valoare, sintaxa instrucţiunii *Declare* este:

```
[Public | Private] Declare Function denumireaFuncţiei _
Lib "numeleBibliotecii"[Alias "numeleDinDLLAlFunctiei"] _
[([argumente])] [As tip]
```

iar dacă funcţia nu returnează nici o valoare:

```
[Public | Private] Declare Sub denumireaFunctiei _
Lib "numeleBibliotecii" [Alias "numeleDinDLLAlFunctiei"] _
[([argumente])]
```

Domeniul de vizibilitate al funcţiei API

Ca şi în cazul unei proceduri VBA obişnuite, putem stabili vizibilitatea funcţiilor declarate prin intermediul instrucţiunii *Declare*. Astfel, dacă instrucţiunea *Declare* e precedată de cuvântul cheie *Private*, funcţia API nu va putea fi apelată decât în modulul în care a fost declarată. Dacă folosim cuvântul cheie Public (care e şi implicit), funcţia va putea fi apelată din orice modul.

Denumirea funcţiei API

In cadrul instrucţiunii *Declare* trebuie să specificăm şi denumirea pe care o vom folosi în cod pentru a apela funcţia API respectivă. Această denumire trebuie să respecte regulile impuse pentru orice procedură VBA. Dacă nu folosim clauza Alias pentru a specifica numele din DLL al funcţiei, denumirea trebuie să fie exact aceeaşi cu acest nume din DLL.

Specificarea bibliotecii DLL

In cadrul clauzei Lib a instrucţiunii *Declare* trebuie să specificăm, între ghilimele, numele bibliotecii DLL şi, eventual, locaţia sa pe disc. Dacă biblioteca DLL este una dintre principalele biblioteci DLL din Windows, putem omite extensia .DLL (de exemplu, "User32", "GDI32" sau "Kernel32"). Dacă nu specificăm calea completă pentru DLL, Windows îl va căuta în ordine:

1. în directorul în care se află Access;
2. în directorul curent;
3. numai pentru Windows NT:în directorul Windows/System32;
4. în directorul Windows/System;
5. în directorul Windows;
6. în directoarele din PATH.

Clauza Alias

Clauza *Alias* a instrucţiunii *Declare* permite să schimbăm numele unei funcţii API din cel specificat în DLL într-unul permis în VBA. Astfel, dacă am specificat o altă denumire pentru funcţie decât cea dată în DLL, trebuie să includem în instrucţiunea Declare şi clauza Alias, care să conţină numele exact al funcţiei.
Exemplu: pentru a declara funcţia API_1write(), trebuie să folosim un alias, deoarece în VBA numele funcţiilor nu pot începe cu caracterul underscore(_):

```
Declare Function 1write Lib "Kernel32" Alias "_1write" _
(ByVal hFile As Integer, ByVal lpBuffer As String, ByVal intBytes _
                                    As Integer) As Integer
```

Astfel, pentru a apela în codul dumneavoastră funcţia _lwrite() din Kernel32.DLL, veţi folosi denumirea 1write().

Argumente

In mod implicit, Access dă unei proceduri, ca argumente, pointeri la adresa de memorie a variabilelor şi nu valorile explicite ale acestora. Cu toate acestea, multe funcţii API aşteaptă să primească drept argumente valorile variabilelor şi nu pointeri la

adresele lor. In acest caz, trebuie să dăm ca argumente, tot valori şi acest lucru îl realizăm plasând cuvântul cheie ByVal în faţa numelor argumentelor funcţiei din cadrul instrucţiunii *Declare:*

```
Declare Function GetSystemMetrics Lib "user32" (ByVal nIndex_As Long) As Long
```

Şiruri de caractere ca argumente

Multe funcţii API primesc ca argumente şiruri de caractere ce se termină cu caracterul '\0' (al cărui cod ASCII este 0). VBA nu lucrează cu astfel de şiruri de caractere. De aceea, pentru a putea da un şir VBA ca argument unei funcţii dintr-un DLL, trebuie întâi să îl transformaţi într-un şir ce se termină cu '\0'. Tot cuvântul cheie ByVal vă ajută şi de această dată. Astfel, dacă folosiţi acest cuvânt cheie pentru un argument de tip String, VBA îl converteşte într-un şir terminat cu '\0' şi apoi îi dă funcţiei API respective un pointer la adresa de memorie a şirului. Deşi acest fapt vine în contradicţie cu ce am spus până acum, el se aplică totuşi şirurilor în VBA.

Dacă o funcţie dintr-un DLL primeşte ca argument valoarea unei variabile, ea nu va putea modifica efectiv această valoare. Dacă însă funcţia primeşte adresa de memorie a variabilelor, atunci ea îi poate modifica valoarea. Cum aceasta se întâmplă şi în cazul şirurilor de caractere, pot apărea probleme. Dacă funcţia din DLL modifică valoarea şirului de caractere primit ca argument şi noua valoare conţine mai multe caractere, funcţia nu modifică şi dimensiunea şirului. Ea scrie noul şir la adresa respectivă, suprascriind caracterul '\0' şi ocupând în continuare locaţii de memorie adiacente. Acest fapt poate duce la blocarea aplicaţiei şi chiar a sistemului. Pentru a-l evita, trebuie să vă asiguraţi că şirul pe care-l daţi funcţiei ca argument este suficient de mare pentru a putea păstra valorile pe care aceasta i le va atribui. Puteţi folosi în acest scop un şir de caractere de lungime fixă, suficient de mare:

```
Dim strSir As String* 255
```

Matrice ca argument

Puteţi da ca argumente unei funcţii API elementele unei matrici aşa cum i-aţi da orice altă variabilă. Pentru a face acest lucru, pur şi simplu daţi funcţiei ca argument primul element al matricei. Funcţia va şti să regăsească şi celelalte elemente. Deoarece elementele unei matrice se află la locaţii consecutive de memorie, este suficient ca funcţia să ştie unde începe matricea şi dimensiunea ei. Nu puteţi da ca argument unei funcţii dintr-un DLL decât matrice numerice.

Transformarea tipurilor de date ale argumentelor

Deoarece majoritatea bibliotecilor DLL sunt scrise în C sau C++, tipurile de date ale argumentelor funcţiilor pe care acestea le conţin nu sunt aceleaşi cu tipurile de date VBA. De aceea, e necesar să folosim în instrucţiunea *Declare* tipul de date VBA corespunzător tipului C al argumentului funcţiei din DLL. Tabelul următor arată corespondenţele dintre tipurile de date din limbajul C şi tipurile VBA.

Tip de date în C	Corespondent în VBA
ATOM, BOOL, HFILE, int, UINT, WORD, WPARAM	ByVal i As Integer
intFar*,UINT FAR*	i As Integer
BYTE	ByVal byt As Byte
BYTE*	Byt As Byte
CALLBACK, DWORD, FARPROC, HACCEL, HANDLE, HBITMAP, HBRUSH, HGLOBAL, HICON, HINSTANCE, HLOCAL, HMENU, HMETAFILE, HMODULE, HPALETTE, HPEN, HRGN, HRSRC, HTASK, HWND, LONG, LPARAM, LRESULT	ByVal adr As Long
char*, LPSTR, LPCSTR	ByVal str As String

Tipuri-utilizator

Multe funcţii API primesc argumente ce au ca tip de date alte tipuri decât cele predefinite. Acestea se numesc tipuri de date definite de utilizator sau tipuri-utilizator. Un tip-utilizator poate fi o structură în C, adică un mod de a grupa variabile de tipuri diferite şi care împreună definesc un nou concept. Una dintre structurile cel mai des folosite în bibliotecile DLL este structura RECT, ce reprezintă un dreptunghi prin laturile sale faţă de marginea din stânga şi, respectiv, de sus a ecranului.

```
Type RECT
      left As Long
      top As Long
      right As Long
      bottom As Long
End Type
```

Este tipul VBA corespunzător structurii RECT în C.
Argumentele ce au ca tip de date un tip-utilizator sunt date prin referinţă (adică prin intermediul unui pointer la adresa din memorie a unei variabile de acel tip).

Pointeri nuli

Există şi cazuri când o funcţie API aşteaptă ca argument pointerul nul, care în VBA este dat ca: `ByVal 0&`

Caracterul & arată că pointerul este pe 32 de biţi (adică Long). Pointerul nul trebuie dat ca argument prin valoare.

Exemplu: funcţia API ClipCursor (pe care o vom folosi în *Exemplul 2*) are argumentul: `lpRect As Any`
Dacă ea primeşte ca argument un pointer la o variabilă de tip RECT, atunci cursorul va fi capturat în dreptunghiul definit de acea variabilă. Dacă primeşte ca argument pointerul nul, ea eliberează cursorul.
Tipul *Any* este folosit pentru a dezactiva mecanismul Access de verificare a corespondenţei dintre tipurile argumentelor din declaraţia unei funcţii şi tipurile argumentelor efective, astfel încât funcţia poate primi orice tip de date pentru un argument de tip Any.

Notă: Atunci când lucrăm cu funcţii DLL, este bine să facem regulat copii ale bazei de date. Este de asemenea bine să salvăm orice alte informaţii din alte aplicaţii Windows deschise, deoarece este posibil ca, din cauza unor erori generate de aceste funcţii, sistemul să se blocheze şi să pierdeţi informaţiile nesalvate.

2.2. Exemple de folosire a funcţiilor API

Exemplul 1:Argumente de tip matrice

Dorim să schimbăm câteva dintre culorile pe care Windows le foloseşte pentru orice aplicaţie: culoarea cu care apare scrisă denumirea aplicaţiei în bara de titlu, culoarea folosită pentru textul butoanelor şi culoarea folosită pentru a desena umbra butoanelor.

1. Creăm formularul Culori, ce conţine două butoane: “Schimbă culori” şi “Refă culori”. In pagina de proprietăţi a formularului efectuăm modificările.
2. In pagina *Event* (evenimente) a ferestrei de proprietăţi a butonului “Schimbă culori” apăsaţi butonul (…) din dreptul câmpului *OnClick*; alegem opţiunea Code Builder şi apăsaţi OK, pentru a deschide modulul formularului şi a edita codul pentru tratarea evenimentului InClick al acestui buton.
3. La secţiunea (General) (Declarations) a modulului adăugăm următoarele linii de cod:

```
Const COLOR_BTNTE_T = 18
Const COLOR_CAPTIONTE_T = 9
Const COLOR_BTNS_ADOW_T = 16
Private Declare Function GetSysColor Lib "user32" (ByVal nIndex As Long) As Long
```

Am declarat trei constante, două funcţii API şi o matrice. Funcţia *GetSysColor* returnează culoarea folosită pentru a desena un anumit element al interfeţei cu utilizatorul. Ea primeşte ca argument o valoare ce defineşte culoarea elementului respectiv. Valoarea COLOR_BTNTE T reprezintă culoarea textului butoanelor, COLOR_CAPTIONTE T reprezintă culoarea denumirii aplicaţiilor, iar COLOR_BTNS ADOW reprezintă culoarea cu care se desenează umbrele butoanelor. Funcţia *SetSysColors* modifică culorile anumitor elemente ale interfeţei cu utilizatorul. Ea va primi ca argument numărul elementelor ce urmează să-şi modifice culoarea, o matrice ce conţine aceste elemente şi o matrice cu noile culori.

4. Observăm că în modul există deja o definiţie vidă a procedurii pentru tratarea evenimentului Click al butonului “Schimbă culori”. In corpul acestei proceduri introducem următoarele linii de cod:

```
Dim lTextButon As Long
Dim lTextTitlu As Long
Dim lTextMeniu As Long
Dim elem(2) As Long
elem(0) = COLOR_BTNTE_T
elem(1) = COLOR_CAPTIONTE_T
elem(2) = COLOR_BTNS_ADOW_T
lTextButon = GetSysColor(elem(0))
lTextTitlu = GetSysColor(elem(1))
lTextMeniu = GetSysColor(elem(2))
culori(0) = RGB(255, 0, 0) 'roşu
culori(1) = RGB(0, 155, 0) 'verde
culori(2) = RGB(0, 0, 255) 'albastru
Call SetSysColors(3, elem(0), culori(0))
Culori(0) = lTextButon
```

```
Culori(1) = 1TextTitlu
Culori(2) = 1TextMeniu
```

Această procedură obţine întâi culorile iniţiale ale elementelor specificate anterior. Funcţia RGB creează o culoare, date fiind cantităţile de roşu, verde şi albastru (între 0 şi 255) ce o compun. Matricea culori, care este publică în cadrul modulului, primeşte iniţial ca valori ale elementelor culorile roşu, verde şi albastru pe care le vom folosi ca noi culori pentru elementele grafice specificate. Apelând funcţia *SetSysColors*, facem efectiv modificările, după care, în matricea culorii păstrăm vechile culori obţinute cu ajutorul funcţiei *GetSysColor* (pentru a le putea reface ulterior).

5. Pentru a scrie o procedură pentru tratarea evenimentului *OnClick* al butonului "Refă culori", dechidem pagina de proprietăţi. In modulul formularului va apărea o procedură cu corpul vid. Scrieţi în corpul ei următoarele linii de cod:

```
Din elem(2) As Long
elem(0) = COLOR_BTNTE_T
elem(1) = COLOR_CAPTIONTE T
elem(2) = COLOR_BTNS_ADOW_T
Call SetSysColors(3, elem(0), culori(0))
```

Butonul "Refă culori" va da elementelor grafice specificate vechile culori şi anume cele stocate în matricea culori (care are ca domeniu de vizibilitate întregul modulul).

6. Trecem în modul Form View şi testăm funcţionarea celor două butoane.

Exemplul 2: Lucrul cu pointerul nul

Mouse-ul este pentru dumneavoastră un mijloc indispensabil de a interacţiona cu o aplicaţie. Ce-aţi spune dacă nu aţi putea muta cursorul mouse-ului decât într-o anumită porţiune a ecranului? ☺

1. Creăm formularul Cursor ce conţine două butoane: Capturează şi Eliberează. In pagina sa de proprietăţi efectuăm modificările.
2. La secţiunea (General) (Declarations) a modulului formularului adăugăm următoarele declaraţii:

```
Private Type RECT
      Left As Long
      Top As Long
      Right As Long
      Botton As Long
End Type
Private Declare Function GetWindowsRect Lib "user32" (ByVal hwnd_
   As Long, lpRect As RECT) As Long
Private Declare Function ClipCursor Lib "user32" (lpRect As Any)_
   As Long
```

Funcţia FetWindowsRect returnează dreptunghiul ocupat de o anumită fereastră.

3. In cadrul corpului procedurii pentru tratarea evenimentului OnClick al butonului Capturează, scrieţi următoarele linii de cod:

```
Dim rectForm As RECT
Call GetWindowRect(Me.hwnd, rectForm)
Call ClipCursor(rectForm)
```

Apăsând pe butonul Capturează, cursorul mouse-ului se va mai putea mişca doar în interiorul formularului Cursor. Funcţia GETWindowRect primeşte ca argument identificarea formularului (Me.hwnd) şi adresa unei variabile de tip RECT (rectForm) în care va scrie coordonatele dreptunghiului ce mărgineşte formularul. Apoi, dăm acest dreptunghi (mai precis adresa lui) ca argument funcţiei ClipCursor, care va imobiliza cursorul mouse-ului în interiorul formularului.

4. In corpul procedurii de tratare a evenimentului ObClick al butonului Eliberează, scriem:

```
Call ClipCursor(ByVal 0&)
```

5. Treceţi în modul Form View şi testaţi funcţionarea celor două butoane. Observaţi atunci când cursorul mouse-ului este capturat, aveţi totuşi acces la bara de titlu a formularului (deci şi la meniul sistem şi la butoanele sistem ale acestuia). Dacă nu doriţi acest lucru, puteţi micşora corespunzător dreptunghiul în care se poate mişca mouse-ul (prin intermediul elementelor variabilei RectForm).