# Curs 8. Visual Basic for Applications (VBA)



Limbajul VBA (Visual Basic for Applications) este un limbaj de programare complex, ce permite dezvoltarea de aplicaţii atât în Access, cât şi în Word, Excel, Microsoft Project şi Microsoft Visual Basic. Avantajele programării în VBA: interceptarea şi tratarea erorilor, un control mai bun asupra interfeţei cu utilizatorul, mai multă rapiditate în execuţie, posibilitatea de a interacţiona cu alte aplicaţii şi de a apela funcţii Windows.

## 1. Mediul de programare Access

Access păstrează codul procedurilor scrise în *module*. Acestea se impart în trei categorii: module standard, module pentru clase şi module pentru formulare şi rapoarte.
Modulele standard sunt obiecte de sine stătătoare şi sunt folosite pentru a crea şi stoca proceduri ce nu sunt legate de un anumit formular sau raport.
Modulele pentru clase sunt folosite pentru a crea şi a defini obiecte cărora le puteţi apoi crea instanţe. Modulele pentru formulare şi rapoarte se deosebesc de celelalte tipuri de module prin faptul că nu sunt obiecte independente, ci sunt incluse în formularul sau raportul corespunzător. Ele conţin procedurile pentru tratarea evenimentului formularului sau raportului respectiv şi pot fi deschise cu ajutorul optiunii *Database Tools.*

### Conţinutul modulelor

Atunci când un modul este deschis, în partea de sus a ferestrei asociate vom vedea două liste derulante. Acestea ajută să găsim diferitele proceduri stocate în modulul respectiv. Cel din stânga permite să alegem obiectul al cărui cod dorim să-l vedem sau să-l editäm. In cazul modulelor standard, singura opţiune este *General*, deoarece acestea nu conţin obiecte, în timp ce în cazul unui modul al unui formular sau raport, această casetă combinată va afişa lista tuturor obiectelor pe care le conţine. După ce am selectat un obiect, în lista din dreapta vom putea alege un eveniment al acelui obiect.

### (General)

*General* nu este un obiect, ci o secţiune în cadrul unui modul. Aici vom putea pune tot ce ţine de modulul respectiv, în general: opţiuni, declaraţii, proceduri are nu tratează evenimente (în cazul modulelor standard, toate procedurile sunt aici).

### Opţiuni

Orice modul are un set de opţiuni, pe care le putem stabili cu ajutorul cuvântului cheie *Option* şi care sunt prezentate în continuare:

1. **Option Base** – permite să stabilim indicele cel mai mic pentru elementele unei matrice (array). Implicit, acesta este zero.
2. **Option Compare** – determină modul în care Access va compara două şiruri de caractere (lucru util, de exemplu, pentru sortări).
    - **Option Compare Binary** – Access va folosi reprezentarea binară a caracterelor pentru comparaţiile făcute în cadrul modulului;
    - **Option Compare Database** – opţiune implicită ce face ca Access să folosească ordinea de sortare a bazei de date pentru efectuarea comparaţiilor între şiruri de caractere. Această ordine poate fi schimbată cu ajutorul meniului *Tools* | *Options*, dacă în pagina *General* a cutiei de dialog ce se va deschide vom selecta o altă valoare în caseta combinată *New Database Sort Order*.
    - **Option Compare Text** – similară cu *Option Compare Database*, cu deosebirea că la efectuarea comparaţiilor între şiruri de caractere nu se face diferenţă între literele mari şi mici.
3. **Option Explicit** – Dacă specificăm această opţiune, toate variabilele vor trebui să fie declarate înainte de a fi folosite în cadrul modulului.
4. **Option Private** – Dacă stabilim această opţiune, codul aflat în interiorul modulului nu va putea fi accesat dintr-un alt modul.

**Declaraţii (Declarations)** La secţiunea (*General)* a unui modul putem declara proceduri, variabile şi constante ce pot fi folosite în întregul modul.

## Proceduri pentru tratarea evenimentelor

Modulele pentru formulare şi rapoarte pot conţine şi alte secţiuni, în afară de (*General)*. Acestea corespund controalelor şi secţiunilor formularului sau raportului respectiv şi conţin procedurile pentru tratarea evenimentelor legate între ele.

# 2. Proceduri

O *procedură* este o înşiruire de linii de cod, care are ca scop îndeplinirea unui anumit obiectiv. In Access, procedurile pot fi de două tipuri: *subrutine* şi *funcţii*.
O subrutină este un bloc de cod care are un nume. Blocul de cod ce compune subrutina poate fi folosit prin apelarea numelui.
O funcţie este ceva asemănător unei subrutine, cu diferenţa că ea returnează o valoare.
Programatorul poate comunica informaţii prin proceduri prin intermediul argumentelor. Acestea sunt variabile ale căror valori au fost deja stabilite şi care transmit procedurilor datele necesare efectuării operaţiilor specifice.
O procedură este un mod de a aduna mai multe linii de cod laolaltă, astfel încât de câte ori veom dori să efectuăm o anumită operaţie, să nu trebuiescă să scriem decât o singură linie de cod, conţinând numele procedurii.

## Declararea procedurilor

Declararea unei proceduri trebuie să conţină următoarele informaţii: tipul procedurii (subrutină sau funcţie), numele procedurii, numele şi tipul argumentelor (dacă acestea există, tipul informaţiei returnate (dacă procedura este o funcţie).

*Exemplu:* Declararea unei subrutine care tipăreşte un şir de caractere.

```
Sub Tiparire (strText As String)
      Debug. Print strText
End Sub
```

Numele subrutinei este Tipărire şi ea are un singur argument, strText, de tip String (şir de caractere). Subrutina va putea fi apelată astfel: `Call Tiparire ("Salut!")`

*Exemplu:* Declararea unei furncţii care returnează rezultatul împărţirii a două numere reale.

```
Function Cat (db1Nr1 As Double, db1Nr2 As Double) As Double
      If dbNr2 = 0 Then Cat = 0
         Else Cat = db1Nr1 / db1Nr2
      End If
End Function
```

Pentru a apela funcţia, vom scrie: `Db1Rez = Cat (n1, n2)`

unde n1 şi n2 sunt variabile ale căror valori au fost stabilite anterior. Pentru ca rezultatul să fie correct, este importantă ordinea în care sunt date argumentele. Există însă şi o modalitate de a furniza argumentele unei funcţii în altă ordine decât cea în care acestea au fost date la declararea funcţiei: în faţa argumentului efectiv, specificaţi numele argumentului din declaraţie, urmat de două puncte şi egal, ca şi în exemplul de mai jos.

```
db1Rez = Cat (db1Nr2: = n2, db1Nr1:= n1)
```

In general, pentru ca apelul unei proceduri să se poată compila, trebuie să fie furnizate toate argumentele din declaraţie. Putem însă modifica acest comportament, dacă în cadrul declaraţiei vom specifica înaintea numelui unui argument cuvântul cheie *Optional*. Astfel, la apelarea procedurii, nu va mai trebui neaparat să daţi valoarea efectivă a acelui argument.

*Exemplu:* funcţia Cat poate fi declarată ca:

```
Function cat (Optional db1Nr1 As Double, db1Nr2 As Double) As Double
```

şi apoi apelată astfel: `db1Rez = Cat(, n2)` sau `db1Rez = Cat (db1Nr2:= n2)`

Atunci când apelăm o procedură fără să furnizăm unele dintre argumentele opţionale, trebuie ca în locul lor să punem o virgulă sau să specificăm pentru argumentele neopţionale, numele acestora din declaraţie. Este bine ca la declararea unei proceduri să lăsăm la urmă argumentele opţionale, pentru ca la apelare să nu mai trebuiască să includeţi virgule sau nume. Astfel putem să declarăm funcţia Cat aşa:

```
Function Cat (db1Nr2 As Double, Optional db1Nr1 As Double) As Double
```

şi apoi apelată astfel: `db1Rez = Cat (n2)`

**Notă:** Atât folosirea argumentelor opţionale cât şi a celor complet specificate (cu ajutorul numelui din declaraţie) este recomandată mai ales atunci când procedura are un număr mare de argumente.

## Lucrul cu argumente opţionale

Putem afla dacă un argument declarat ca opţional a fost sau nu furnizat la apelul procedurii; Pentru aceasta există o funcţie Access predefinită, *IsMissing*, care returnează valoarea *True* dacă argumentul lipseşte şi *False* dacă acesta a fost furnizat. Putem rescrie funcţia Cat astfel încât, dacă deîmpărţitul lipseşte, să returneze valoarea zero:

```
Function Cat(db1Nr2 As Double, Optional db1Nr1 As Double) As Double
      If IsMissing(db1Nr1) Then Cat = 0
        Else If db1Nr2 = 0 Then Cat = 0
             Else Cat = db1Nr1 / db1Nr2
             End If
      End If
End Function
```

## Funcţii predefinite

*IsMissing* este una dintre funcţiile predefinite pe care Access vi le pune la dispoziţie. Există funcţii predefinite pentru toate tipurile de activităţi: lucrul cu test şi date numerice, comunicarea cu alte aplicaţii, lucrul cu fişiere şi directoare, cu date şi ore etc; Le găsim în *Expression Builder*.

# 3. Variabile

Variabilele sunt utilizate pentru a stoca date folosite în subrutine sau funcţii (asemeni unor containere). Putem atribui o valoare unei variabile şi apoi să efectuăm calcule cu valoarea respectivă prin intermediul variabilei. Folosind variabila şi nu direct valoarea, dăm flexibilitate codului pe care îl vom putea apoi folosi şi pentru alte valori.
Variabilele sunt nişte "containere" folosite pentru a stoca date. În Access datele sunt stocate în tabele. Spre deosebire de tabele, variabilele sunt folosite pentru a stoca o informaţie temporară, ce ne ajută să obţinem un anumit rezultat. Tabelele sunt folosite pentru a stoca informaţii pentru o perioadă mai lungă de timp.

## Tipuri de date

Tipurile de date sunt folosite pentru a specifica ce fel de informaţie va fi păstrată într-o variabilă astfel încât Access să ştie cum să stocheze şi cum să manipuleze această informaţie.

| Tipul de date | Domeniul | Exemplu |
|---|---|---|
| String | şir de caractere | Dim strText As String<br>Dim strTextS |
| Integer | numere întregi între -32768 şi 32767 | Dim intNr As Integer<br>Dim intNr% |
| Long | numere întregi între -2147483648 şi 2147483647 | Dim intNr As Long<br>Dim IngNr& |
| Single | numere reale positive sau negative cu maximum 7 zecimale, cu valoare absolută între 1.401298*10 şi 3.402823*$10^{38}$ | Dim sngNr As Single<br>Dim sngNr! |
| Double | Numere reale positive sau negative cu maximum 15 zecimale, cu valoare absolută între 4.94065645842147*$10^{-324}$ şi 1.79769313486231*$10^{308}$ | Dim db1Nr As Double<br>Dim db1Nr# |
| Currency | Număr cu 4 zecimale între -922337203685477.5808 şi 922337203685477.5807, folosite mai ales în domeniul bancar | Dim crnNr@ |
| Boolean | Pentru variabile care pot avea numai valoarea True sau Flase | Dim bNr As Boolean |
| Date | Date între 1 Jan 100 şi 31 Dec 9999 şi ore între 00:00:00 şi 23:59:59 | Dim datData As Date |
| Byte | Valori întregi între 0 şi 255 | Dim byNr As Byte |
| Object | Orice tip de obiecte, nu numai obiecte Access 97 | Dim objObiect As Obiect |
| Variant | Folosite pentru a stoca diferite tipuri de date | Dim varOrice As Variant |
| Hyperlink | Stochează date de tip text ce reprezintă adrese de hiperlegături | Dim hypPag As Hyperlink |

## Declararea variabilelor

Dacă la secţiunea (General)(Declaration) a unui modul am stabilit opţiunea *Option Explicit*, înainte de a putea atribui o valoare unei variabile, trebuie să indicaţi: numele variabilei şi tipul datelor pe care acesta le va stoca; cu alte cuvinte, trebuie să declaraţi variabila. Acest lucru îl veţi face prin intermediul instrucţiunii `Dim` şi a clauzei `As Type`.

```
Dim intNumar As Integer
```

Prin linia de cod de mai sus am declarat variabila intNumar de tip *Integer*. Atunci când alegem numele pentru o variabilă, trebuie să ţinem cont de următoarele reguli:

- Trebuie să înceapă cu o literă
- Poate să conţină doar litere, cifre şi caracterul underscore ( _ )
- Poate avea cel mult 40 de caractere
- Nu poate fi un cuvânt rezervat.

Dacă nu am hotărât de la început ce tip de date va stoca variabila o putem face ulterior folosind tipul de date *Variant.*

**Tipul Variant** Intr-o variabilă de acest tip putem stoca text, date numerice sau orice alt tip de date predefinit în Access.

*Exemplu:*

```
Dim varOrice As Variant
varOrice = 32
varOrice = varOrice & " este un număr întreg"
```

In prima linie de cod declarăm variabila varOrice ca fiind de tipul Variant, apoi îi atribuim valoarea 32. In cea de-a 3-a linie de cod am folosite caracterul & pentru a concatena 32 cu un şir de caractere. Astfel, variabila noastră conţine şitul de caractere "32 este un număr întreg". Access a făcut implicit conversia de tip întreg la şir de caractere.

*Exemplu:*

```
varOrice = "2.5"
varOrice = varOrice + 10
```

Cea de-a doua linie de cod va face ca valoarea lui VarOrice să fie 12.5, prin conversia şirului de caractere "2.5" la numărul real 2.5. Acelaşi efect îl vor avea şi următoarele linii de cod:

```
varOrice = 2.5
varOrice = varOrice + "10"
```

Şirul "10" va fi convertit la numărul întreg 10.

Atunci când lucrăm cu variabile de tipul Variant, trebuie să ţinem cont de următoarele:

- Access va converti variabila la tipul de date corespunzător valorii atribuite;
- Atunci când doriţi să concatenaţi două şiruri de caractere, folosiţi operatorul & în loc de +;

Când folosiţi operatorul + pentru lucrul cu variabile de tip Variant, Access va acţiona astfel:

- Dacă ambele valori sunt numerice, se va efectua o adunare;
- Dacă ambele conţin şiruri de caractere, ele vor fi concatenate;
- Dacă una este o valoare numerică şi cealaltă este un şir de caractere, Access va încerca s-o convertească în valoare numerică şi să facă adunarea. Dacă nu reuşeşte, va rezulta o eroare.

Pentru a determina ce tip de date conţine o variabilă de tip Variant la un moment dat, apelaţi funcţia predefinită VarType. Aceasta va returna una dintre valorile prezentate în tabelul următor.

| Valoarea returnată | Tipul de date | Constanta intrinsecă |
|---|---|---|
| 0 | Vid (variabila este neiniţializată) | vbEmpty |
| 1 | Null | vbNull |
| 2 | Integer | vbInteger |
| 3 | Long | VbLong |
| 4 | Single | vbSingle |
| 5 | Double | vbDouble |
| 6 | Currency | vbCurrency |
| 7 | Date | vbDate |
| 8 | String | vbString |
| 9 | Object | vbObject |

| | | |
|---|---|---|
| 10 | Eroare | vbError |
| 11 | Boolean | vbBoolean |
| 12 | Variant (numai pentru matrice) | vbVariant |
| 13 | Obiect DAO | vbDataObject |
| 14 | Decimal | vbDecimal |
| 17 | Byte | vbByte |
| 8192 | matrice | vbArray |

**Notă:** In ultima coloană a tabelului am specificat constanta intrinsecă ce corespunde valorii returnate.

Atunci când o variabilă este declarată, ei i se atribuie valoarea iniţială 0 dacă variabila e de tip numeric sau " " (şirul vid) dacă e de tip şir de caractere. Variabilelor de tip Variant însă, nu li se atribuie la declarare nici o valoare, cu alte cuvinte sunt "vide' (a nu se confunda aceasta cu valoarea 0 sau " " şi nici cu valoarea Null). Aceste variabile rămân vide până în momentul când li se atribuie o valoare. Puteţi testa dacă o variabilă de tip Variant a fost sau nu iniţializată cu ajutorul funcţiei IsEmpty, care returnează valoarea True dacă variabila este "vidă". Pentru a "vida" din nou o variabilă, trebuie să-i atribuiţi ca valoare o altă variabilă de tip Variant care nu a fost încă iniţializată.

```
Dim varOrice As Variant
' acum vorOrice este vida
varOrice = 0
If IsEmpty(varOrice) Then …
' am testat daca varOrice este vida (si nu este)
Dim VarOrice As Variant
varOrice = varCeva
' acum varOrice este vida din nou
```

**Notă:** In Access, caracterul apostrof (') introduce un comentariu într-un modul.

O altă valoare specială pe care o poate avea o variabilă de tip Variant este Null (dacă veţi încerca să atribuiţi valoarea Null unei variabile de orice alt tip decât Variant, veţi obţine o eroare). Dacă folosiţi valoarea Null într-o expresie, întreaga expresie va avea valoarea Null.
*Exemplu:*

```
Dim varOrice As Variant
varOrice = Null
varOrice = varOrice + 2.5
```

După cea de-a treia linie de cod, varOrice va avea tot valoarea Null. Dacă înlocuiţi această linie cu:

```
varOrice = varOrice & "ceva"
```

varOrice va avea valoarea "ceva". Deci, operatorul & este singurul care nu propagă valoarea Null într-o expresie.
Puteţi testa dacă o variabilă are valoarea Null cu ajutorul funcţiei IsNull.

Variabilele de tip Variant sunt foarte utile, deoarece, pe lângă faptul că nu trebuie să alegem un tip de date, sunt singurul tip de variabile cărora li se poate atribui valoarea Null. Astfel, dacă unei anumite variabile îi veţi atribui valori dintr-un camp al unei tabele care poate conţine şi valoarea Null, acea variabilă trebuie să fie de tip Variant. Folosirea acestui tip de variabile are însă şi dezavantaje, printre care încetinirea execuţiei aplicaţiei şi chiar posibila apariţie a unor erori.

## Option Explicit

Să revenim acum la Opţiunea Explicit şi să vedem ce se poate întâmpla în cazul în care nu am stabilit această opţiune în cadrul secţiunii (General) a unui modul. Pentru aceasta, vom considera următoarea subrutină:

```
Sub AriaCercului(db1Raza As Double)
Dim db1Aria As Double
Db1Aria = 3.14 * db1Rsza *db1Raza
Deburg.Print "Aria = " & db1Aria
End Sub
```

Pentru a rula această subrutină, deschidem fereastra *Immediate.* Scriem în partea de jos a ferestrei linia:

```
Call AriaCercului (5)
```

După aceasta, apăsaţi pe Enter şi veţi vedea rezultatul: 0. Este clar că acest rezultat este eronat. Motivul este acela că în codul subrutinei s-a strecurat o greşeală: în loc de db1Raza am scris, într-un loc, db1Rsza. Opţiunea Explicit nefiind stabilită, şi deci nefiind necesar ca variabila să fie declarată înainte de a fi folosită, Access a atribuit variabilei db1Raza valoarea implicită 0, care a dus la acest rezultat eronat.

**Notă:** Pentru ca opţiunea Explicit să fie adăugată automat la orice modul nou pe care îl creaţi, alegeţi meniul *Tools* | *Options* şi, în pagina *Editor* a ferestrei *Options*, validaţi opţiunea *Require Variable Declaration*.

## Constante

Nu le putem schimba valoarea în timpul execuţiei programului. In Access există trei tipuri de constante:

- constante simbolice, create cu ajutorul instrucţiunii Const,
- constante intrinsece,
- constante de sistem: True, False şi Null.

Cu ajutorul constantelor simbolice putem da un nume unei valori. Astfel, codul va fi mai uşor de citit şi de întreţinut.
*Exemplu:*

```
Const csPi = 3.14
```

am definit constanta csPi, a cărei valoare este 3.14. Constantele pot fi folosite oriunde pot fi folosite şi variabilele. Mai mult, ele pot fi folosite la definirea altor constante:

```
Const csOra  = csMinut *60
```

Putem atribui şi tipuri de date constantelor, astfel încât Access să le poată recunoaşte şi să putem efectua operaţii cu ele:

```
Const csPi As Double = 3.14
```

Constantele intrinsece sunt constante definite în Access. Astfel, putem scrie în codul nostru ori:

```
If VarType(avrOrice) = 2 Then_
```

ori:

```
If VarType(avrOrice) = vbInteger Then_
```

## Vizibilitatea şi durata de viaţă a variabilelor

Vizibilitatea unei variabile este dată de porţiunea de cod în care ea poate fi folosită şi depinde de locul în care a fost declarată. Vizibilitatea unei variabile create în cadrul unei proceduri se reduce doar la acea procedură. O variabilă poate fi declarată şi în afara oricărei proceduri, în cadrul secţiunii (General) (Declarations) a unui modul. In acest caz, ea poate fi folosită de către toate procedurile din modulul respective (şi, uneori, şi din alte module) şi spunem că vizibilitatea sa este publică.

Durata de viaţă a unei variabile e dată de perioada în care ea va putea conţine o valoare. Astfel, o variabilă a unei proceduri "trăieşte" atâta timp cât se rulează procedura respectivă şi este recreată la următorul apel al procedurii. Dacă doriţi ca o variabilă locală să existe şi după terminarea procedurii, folosiţi cuvântul cheie Static la declararea ei. Puteţi declara şi o procedură ca find statică, astfel încăt toate variabilele sale locale să fie statice.
Variabilele publice există atâta timp cât este deschisă baza de date.
Putem stabili şi vizibilitatea procedurilor, folosind cuvintele cheie *Public* şi *Private*. O procedură având specificatorul *Private* poate fi apelată numai în interiorul modulului în care este declarată, în timp ce o procedură cu specificatorul *Public* poate fi apelată din orice alt modul. Pentru a putea apela o procedură în fereastra *Immediate*, aceasta trebuie să fie publică.
In exemplele de proceduri pe care le-am prezentat până acum, toate variabilele erau locale şi deci valorile lor se pierdeau după terminarea rulării procedurilor în care erau declarate.

## Variabile statice

Pentru ca o variabilă declarată într-o procedură să-şi păstreze valoarea şi între două apeluri ale procedurii, ea trebuie să fie declarată folosind cuvântul cheie Static, în locul lui Dim. O astfel de variabilă va fi iniţializată o singură dată: la primul apel al procedurii.
*Exemplu:*

```
Sub VarStatic ()
Static intVar AsInteger
intVar = intVar + 1
Debug.Print intVar
End Sub
```

Rulaţi subrutina de mai multe ori.

Dacă intVar nu ar fi fost declarată ca statică, subrutina ar fi tipărit de fiecare dată aceeaşi valoare: 1. Puteţi ca toate variabilele unei proceduri să fie statice, folosiţi cuvântul cheie Static înaintea declaraţiei procedurii:

```
Static Sub VarStatic()
```

## Variabile publice

Există două tipuri de variabile publice în Access:

- variabile publice în cadrul unui modul
- variabile publice în cadrul aplicaţiei.

Variabilele publice în cadrul unui modul se declară cu ajutorul instrucţiunii Dim la secţiune (General) (Declarations) a modulului şi pot fi folosite de către toate procedurile modulului. Următorul exemplu foloseşte astfel de variabile.

1. La secţiunea (Declarations) a unui modul declaraţi variabila intVarModul astfel: `Dim int VarModul As Integer`
2. Creaţi următoarele două subrutine:

```
Sub Incrementare ( )
intVarModul = intVarModul + 1
End Sub
```

şi

```
Sub tipărire()
Debug.Print intVarModul
End Sub
```

3. Apelaţi de mai multe ori ambele subrutine din fereastra *Immediate*.
4. Creaţi o a 3-a procedură, în acelaşi modul:

```
Sub LocalVar()
Dim intVarModul As Integer
intVarModul = 50
Debug.Print intVarModulEnd Sub
```

5. Apelaţi acum subrutina *LocalVar* şi apoi, pe *Tipărire.*

LocalVar lucrează cu variabila intVarModul declarată în interiorul său fără a modifica sau a fi afectată de variabila publică intVarModul, chiar dacă acestea au acelaşi nume.

**Notă:** Pentru a evita confuziile este bine să alegem pentru variabilele locale nume diferite de cele ale variabilelor publice.

Variabilele publice în cadrul aplicaţiei pot fi accesate din toate modulele aplicaţiei, dar pot fi declarate numai în cadrul secţiunii (General) (Declarations) a unui modul standard.
Aceste variabile se declară înlocuind instrucţiunea `Dim` cu instrucţiunea `Public:`

```
Public intVarPublic As Integer
```

Nu putem declara variabile publice în cadrul procedurilor.

**Notă:** Folosiţi variabile publice în cadrul aplicaţiei numai dacă este neapărat nevoie. Astfel vă puteţi feri de erori şi codul dumneavoastră va fi mai uşor de întreţinut şi de refolosit. Dacă veţi dori ca mai multe proceduri să folosească o anumită valoare, daţi-o ca parametru sau ca valoare de retur.

# 4. Instrucţiuni de control

VBA vă oferă posibilitatea de a testa în cadrul procedurilor anumite condiţii şi de a efectua operaţii diferite în funcţie de rezultatele testelor. In continuare, vom studia instrucţiunile VBA care vă permit luarea de decizii în urma unor astfel de teste.

## Instrucţiunea If… Then

Vom folosi această instrucţiune în cazul în care vom dori să executăm anumite operaţii doar dacă o condiţie este adevărată. Ea are doar două sintaxe. Dacă instrucţiunea este folosită pe o singură linie de cod, sintaxa este:

```
If conditie Then instructiune
```

Altfel, dacă instrucţiunea e compusă din mai multe linii de cod:

```
If conditie Then
      instructiuni
End If
```

Cea de-a doua variantă este utilă atunci când doriţi să se execute mai multe instrucţiuni în cazul în care condiţia e adevărată.
Exemplu: folosirea ambelor variante ale instrucţiunii *If…Then*

```
Sub TestIfThen(strDat As String, strCautat As String)
Dim strRezultat As String
If strDat = " " Then strRezultat = "Sirul dat este vid"
If InStr(strDat, strCautat) Then
   strRezultat = strCautat
   strCautat = strRezultat & " se află în " & strDat
End If
Debug.Print strRezultat
End Sub
```

In cadrul subrutinei TestIfThen am folosit o funcţie predefinită în Access, InStr, care verifică dacă un şir de caractere se regăseşte în alt şir şi returnează poziţia la care apare şirul căutat în şirul sursă sau 0 dacă şirul căutat nu se regăseşte în şirul sursă.
Orice expresie poate apărea în cadrul clauzei If. Dacă expresia se evaluează la ceva diferit de 0, ea e considerată a fi adevărată, şi falsă dacă valoarea ei e zero.

## Instrucţiunea If… Then…Else

Pentru a specifica ce operaţii trebuie efectuate în cazul în care condiţia e falsă, folosim această instrucţiune.
Putem privi această instrucţiune ca pe un mijloc de a îmbrăca mai multe instrucţiuni *If…The…Else* şi deci, de a verifica mai multe condiţii. *Exemplu:*

```
If strDat = " " Then
   strRezultat = "Sirul dat este vid"
   ElseIf InStr(strDat, strCautat) Then
      strRezultat = strCautat & " se află în " & strDat
      Else
         strRezultat = strCautat & " nu se află în " & strDat
End If
```

Astfel, dacă prima condiţie nu e adevărată, se verifică cea de-a doua şi, dacă nici aceasta nu e adevărată, se execută instrucţiunea din cadrul clauzei Else. In acest mod putem folosi oricâte instrucţiuni EndIf.

## Instrucţiunea Select Case

Atunci când trebuie să testaţi mai multe condiţii şi să efectuaţi pentru fiecare un anumit set de operaţii, folosirea instrucţiunilor ElseIf poate să devină greoaie. Puteţi folosi o instrucţiune *Select Case* care să facă acelaşi lucru. Astfel, două secvenţe de cod duc la acelaşi rezultat:

```
If intVarsta<=18 Then
…
ElseIf intVarsta>=19 And intVarsta<=25 Then
…
ElseIf intVarsta>=26 And intVarsta<=40 Then
…
Else
End If
```

şi

```
Select Case intVarsta
Case Is <=18
…
Case 19 To 25
…
Case 26 To 40
…
Case Else
…
End Select
```

Se poate observa că cea de-a doua variantă este mult mai uşor lizibilă. Expresiile clauzelor Case se testează în ordinea în care sunt date şi, în momentul în care este întâlnită o expresie adevărată, se vor executa instrucţiunile corespunzătoare şi se va ieşi din instrucţiunea Select Case. Astfel, dacă expresiile a două dintre clauzele Case sunt adevărate, vor fi executate numai codul corespunzător primeia dintre ele, nefăcându-se apoi nici o verificare.
Instrucţiunea Select Case de mai sus mai poate fi scrisă şi altfel:

```
Select Case intVarsta
Case Is <=18
…
Case 19, 20, 21, 22, 23, 24, 25
…
Case 26 To 40
…
Case Else
…
End Select
```

Existenţa clauzei Case Else în cadrul unei instrucţiuni Select Case nu este obligator. Ea poate fi însă utilă în cazul în care variabila testată poate avea şi altă valoare decât cele cuprinse în cluzele Case.
**Notă:** Variabila testată poate conţine atât valori numerice, cât şi şiruri de caractere sau date.

## Instrucţiunea IIf (Immediate If)

Această instrucţiune este aproape echivalentă cu instrucţiunea If…Then…Else. Există o diferenţă în modul în care sunt executate cele două instrucţiuni. Sintaxa instrucţiunii IIf:

```
IIf(condiţie, valoarea1, valoarea2),
```

unde:

- condiţie este condiţia care trebuie să fie testată;
- valoarea 1 este valoarea returnată în cazul în care condiţia e adevărată;
- valoarea2 este valoarea returnată când condiţia e falsă.

Să considerăm acum următoarele două funcţii:

```
Function Impartire(intImpartitor As Integer,
intDeimp As Integer) As_ Double
If intImpartitor = 0 Then
Impartire = 0
Else
Imapartire = intDeimp /intImpartitor
End If
End Function
```

şi

```
Function Impartire(intImpartitor As Integer,
intDeimp As Integer) As_ Double
Impartire = IIf(intImpartitor = 0, 0, intDeimp /intImpartitor)
End Function
```

Aparent, cele două funcţii fac acelaşi lucru. Diferenţa constă în faptul că în prima funcţie, dacă este îndeplinită condiţia instrucţiunii If, se execută instrucţiunea Impartire = 0 şi se iese din instrucţiunea If…Then…Else fără alte verificări. In instrucţiunea IIf din cea de-a doua funcţie se evaluează însă toate argumentele. Astfel, dacă intâmplător = 0, la evaluarea lui intDeimp /intImpartitor va rezulta o eroare.
Faptul că în instrucţiunea IIF se evaluează toate argumentele, nu numai că încetineşte execuţia procedurii, dar poate duce şi la erori.

# 5. Instrucţiuni iterative

Există situaţii când un anumit bloc de instrucţiuni trebuie să fie executate de mai multe ori. Pentru astfel de operaţii, VBA vă pune la dispoziţie instrucţiunile *For…Next* şi *Do…Loop*.

## Instrucţiunea For…Next

Această instrucţiune este utilă atunci când ştiţi precis câte iteraţii aveţi de efectuat. Sintaxa:

```
For contor = val_initiala To val_finala [Step pasul]
…
Next [contor]
```

unde:

- contor – este variabila care se incrementează la fiecare iteraţie
- val_initială – valoarea iniţială a variabilei contor
- val_finală – valoarea finală a variabilei contor
- pasul – valoarea cu care se incrementează variabila contor la fiecare iteraţie.

Valoarea implicită a pasului este 1. Pasul poate fi şi negativ, caz în care variabila contor va descreşte la fiecare iteraţie cu valoarea absolută a pasului.

```
For intCont = 1 To intTotal Step2
Debug.Print intCont
Next
```

In exemplul de mai sus, valoarea iniţială a contorului intCont este 1 şi la fiecare pas această valoare creşte cu două unităţi, până când egalează valoarea variabilei intTotal. De asemenea, la fiecare iteraţie se va afişa valoarea lui intCont.

**Instrucţiunea Do…Loop**

Pentru cazurile în care cunoaşteţi dinainte numărul de iteratii ce trebuie efectuate, instrucţiunea *For…Next* este ideală. Sunt însă şi situaţii în care un set de instrucţiuni trebuie să se execute până când sau atâta timp cât o anumită condiţie este satisfăcută. In astfel de cazuri, veţi folosi instrucţiunea *Do…Loop*, care are următoarele sintaxe:

```
Do Until conditie
      bloc de instrucţiuni
Loop
```

sau:

```
Do
      bloc de instructiuni
Loop Until conditie
```

sau

```
Do
      bloc de instructiuni
Loop Until conditie
```

sau

```
Do While codnditie
      bloc de instructiuni
Loop
```

sau

```
Do
      bloc de instructiuni
Loop While conditie
```

Primele variante ale instrucţiunii *Do…Loop* execută blocul de instrucţiuni până când condiţia va deveni adevărată (adică atâta timp cât ea este falsă). Deosebirea dintre ele este aceea că în cea de-a doua variantă blocul de instrucţiuni se execută cel puţin o dată. Ultimele două variante execută blocul de instrucţiuni atâta timp cât condiţia e adevărată, cu deosebirea că, în cazul ultimei variante, blocul de instrucţiuni se execută cel puţin o dată.

## Intreruperea iterării

Atât în cazul instrucţiunii For…Next, cât şi în cel al instrucţiunii *Do…Loop*, ştim când anume se va opri procesul iterativ, VBA vă dă şi posibilitatea de a întrerupe acest proces la o anumită iteraţie, cu ajutorul instrucţiunii *Exit.* Exemplu:

```
For intCont = 1 To intTotal
      If intStop Then
      Exit For
      End If
Debug.Print intCont
Next
```

Dacă intStop este diferit de zero, procesul iterativ se opreşte, indiferent de valoarea lui intCont.
In mod asemănător, pentru a părăsi o subrutină sau o funcţie, putem folosi instrucţiunile *Exit Sub* sau *Exit Function.*