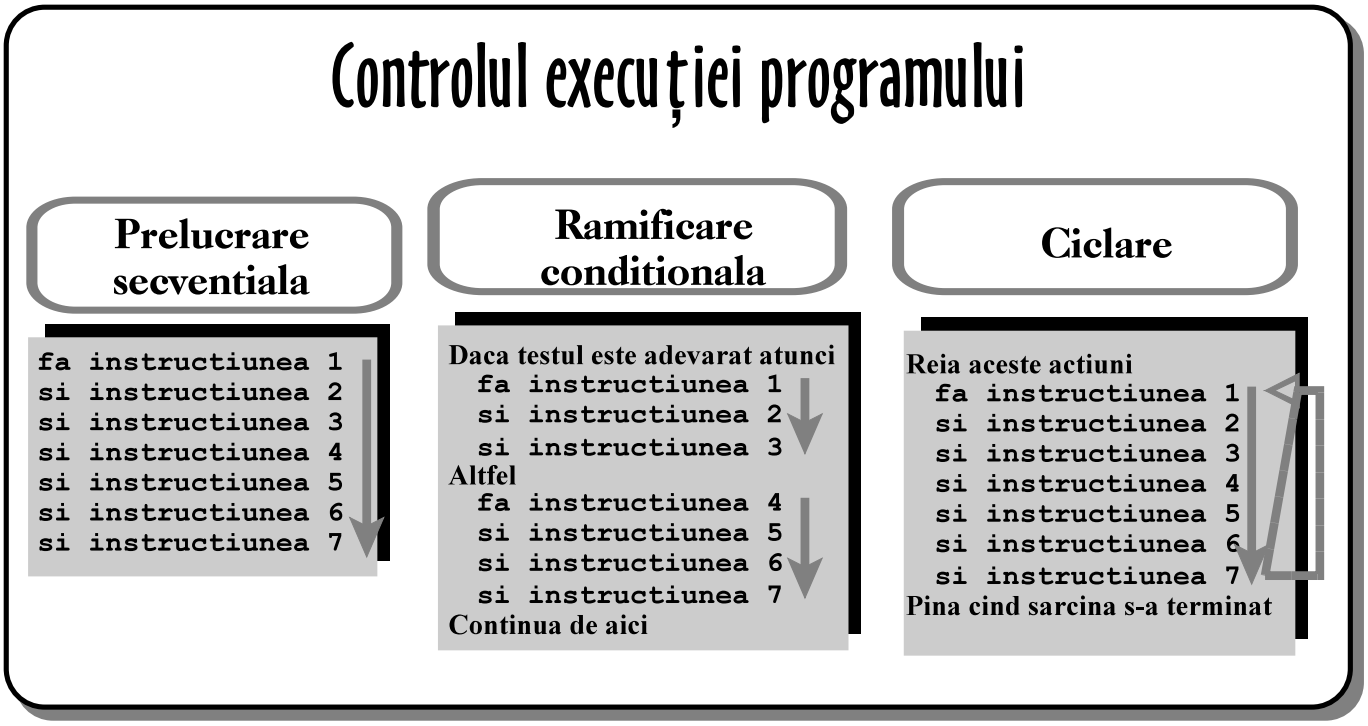
# Cuprins C 3 - VB

# Programare

- **Controlul execuției programului**
- **Cod pentru decizie:**
  `If ... Then ...,Select Case`
- **Cod pentru ciclare:**
  `Do While, For ... Next`
- **Proceduri**
- **Subrutine, Funcții, Vizibilitate**
- **Transferul de parametri**

Toate limbajle de programare au abilitatea de a lua decizii pe baza unor informații, adică să modifice acțiunile pe care urmează să le facă în funcție de rezultele anumitor teste. Limbajele mai au nevoie să poată repeta un grup de instrucțiuni în funcție de anumite criterii variabile.

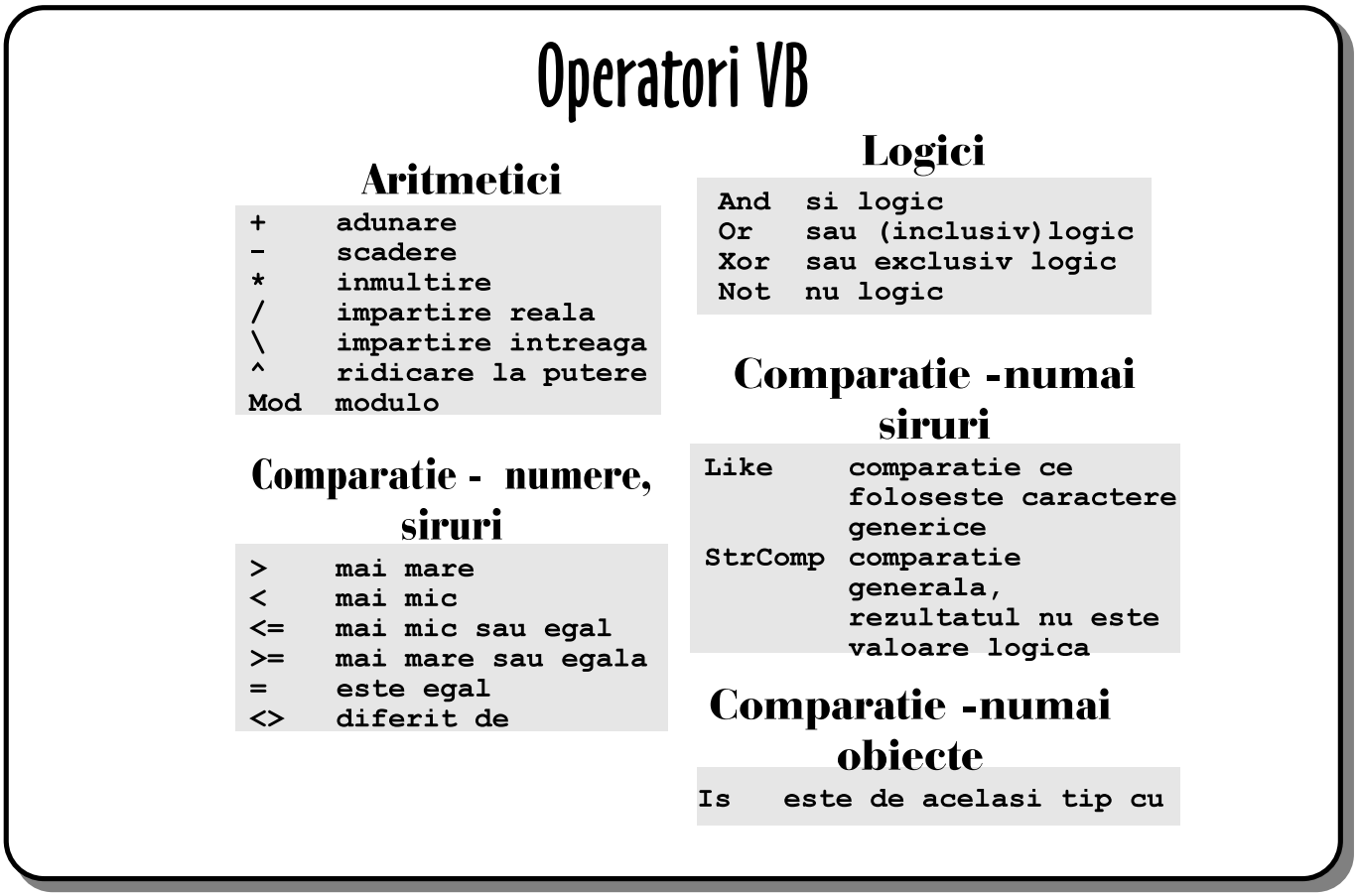
Scrierea secvențelor de cod utilizate frecvent în proceduri pentru ca să poată fi apelate apoi din diferite porțiuni de cod ale aplicației este o altă necesitate fundamentală a tuturor limbajelor. **`Public`** şi **`Private`** permit controlul vizibilității procedurilor, adică a porțiunilor de cod din care acestea pot să fie apelate, iar parametrii permit transferul unor date procedurii respectiv întoarcerea unor rezultate din procedură.




Ordinea naturală de execuţie a instrucţiunilor unui program este de sus în jos, se execută cea mai se sus (prima) instrucţiune, se continuă cu cea de a doua, urmează a treia şi procesul continuă pînă cînd se termină lista instrucţiunilor.

Există cazuri cînd dorim să executăm numai anumite instrucţiuni dacă o anumită condiţie particulară este îndeplinită, de exemplu să validăm un buton de comandă numai dacă anumite cîmpuri au fost completate. Codul trebuie să testeze o condiţie (sau un grup de condiţii) şi apoi să ia decizia corespunzătoare. Dacă condiţia nu este îndeplinită atunci dorim să executăm alte instrucţiuni, de exemplu vom invalida butonul de comandă pentru a face ca starea butonului să fie actualizată la cea a datelor introduse în formular. În VB pentru aceste teste se va folosi instrucţiunea **`If`**, iar în cazul unor ramificări mai complexe **`Select Case`**.

Există şi cazuri în care dorim ca un anumit grup de instrucţiuni să fie executate în mod repetat de un număr fixat de ori sau pînă cînd o condiţie particulară este îndeplinită. VB are ciclurile **`For`** şi **`Do`** pentru obţinerea acestor construcţii de cod.




# 1. Operatori VB

Operatorii au simbolurile prezentate mai sus şi realizează anumite funcţii avînd drept argumente operanzi. Operatorii mai puţin obişnuiţi se prezintă în continuare.

**Operatorii de comparaţie a şirurilor şi numerelor**

Operatorii de comparaţie pentru numere pot fi utilizaţi şi pentru şiruri. Implicit, comparaţiile de şiruri în modulele de cod ţin cont de scrierea cu litere mici sau mari, deci "vasile" nu este egal cu "Vasile". Instrucţiunea opţională **`Option Compare`** poate fi folosită după cum urmează pentru a modifica această setare:

```
Option Compare Text      'nu ţine cont de scrierea cu lit. mari sau mici
Option Compare Binay     'ţine cont de scrierea cu lit. mari sau mici
```

**Operatori specifici comparaţie de şiruri**

Operatorul **`Like`** întoarce rezultat **`Boolean`** şi poate folosi caracterul * pentru a ţine locul unor grupuri de caractere. Şablonul de comparaţie poate conţine următoarele caractere speciale: **`*`** - 0 sau mai multe caractere, **`?`** - un singur caracter, **`#`** - o singură cifră.

```
"Fred Flinstone" Like "F*" ' dă True
"Fred Flinstone" Like "*in*o*" ' dă True
"Fred Flinstone" Like "*in*o" ' dă False
```

**StrComp** întoarec un **Variant** ce indică dacă primul şir este egal cu cel de al doilea (rezultat 0), mai mic decît al doilea (rezultat **-1**) sau mai mare decît al doilea şir (rezultat **1**). Dacă unul dintre şiruri este **Null** întoarce **Null**. Implicit, **StrComp** compară şirurile ţinînd cont de scrierea cu litere mari şi mici, dar comportamentul poate fi modificat printr-un parametru opţional.

```
StrComp("Fred", "fred")                    ' întoarce -1
StrComp("Fred", "Fred")                    ' întoarce 0
StrComp("fred", "Fred")                    ' întoarce 1
StrComp("fred", "Fred", vbTextCompare)     ' întoarce 0
```

## Operatori pentru comparaţia obiectelor

Pentru a putea compara referinţele la două obiecte pentru a verifica dacă referă acelaşi obiect sau nu se foloseşte operatorul

```
frmUltFormular Is frmOptiuni      ' True dacă am avut un Set frmUltFormular =
                                  ' frmOptiuni si nu am modificat valoarea din
                                  ' frmUltFormular
```

## Operatori boleeni

Lucrează numai cu valori boleene (**True** şi **Flase**). Se definiesc folosind tabele de adevăr ce prezintă toate posibilităţile de operanzi asupra căroara poate acţiona.

| And | True | False | Null |
|---|---|---|---|
| **True** | True | False | Null |
| **False** | False | False | False |
| **Null** | Null | False | Null |

| Or | True | False | Null |
|---|---|---|---|
| **True** | True | True | True |
| **False** | True | False | False |
| **Null** | True | Null | Null |

Deseori trebuie să combinăm un număr de condiţii într-o singură condiţie de testat. Cel mai des se folosesc operatorii **And** şi **Or. Not** se foloseşte pentru inversarea unui rezultat de testat. Utilizarea parantezelor este recomandată pentru a defini clar condiţia de testare.

```
strNume="Fred" And strPrenume="Flinstone"
(strNume="Fred") And (strPrenume="Flinstone")
(strNume="Fred") Or (strOras="Berrock")
Not ((strNume="Fred") And (strPrenume="Flinstone"))
```

## Instrucțiunea If

**Testare pentru o singură linie**

```
If Pret > Bani Then NuCumpara
```

**Testare pentru un bloc**

```
If Pret > Bani Then
    NuCumpara
End If
```

```
If Pret > Bani Then
    NuCumpara
Else
    Cumpara
End If
```

# 2. Instrucțiunea de ramificare If

Instrucțiunea **If** apare în majoritatea programelor VB. Ea testează valoarea unei variabile sau expresii care se evaluează pentru a întoarce un rezultat **Boolean**. Dacă rezultatul se evaluează la **True** atunci o instrucțiune sau un grup de instrucțiuni predefinite vor fi executate. Dacă rezultele se evaluează la **False** se poate să existe o instrucțiunie sau un grup de instrucțiuni alternative care vor fi executate.

**Exemplul 1 - test If pentru o singură linie**

```
If Pret > Bani Then NuCumpara
```

Aici testul este expresia **Pret > Bani** care se evaluează. Dacă rezultatul este **True**, atunci instrucțiunea imediat următoare lui **Then** va fi executată, în exemplul de mai sus este vorba despre subrutina cu numele **NuCumpara**. Dacă testul ia valoarea **False** nu se face nici o instrucțiune.

**Exemplul 2 - test If pentru o singură linie cu alternativă**

```
If Pret > Bani Then NuCumpara Else Cumpara
```

Prima oară testul expresia **Pret > Bani** se evaluează. Dacă rezultatul este **True**, atunci instrucțiunea imediat următoare lui **Then** va fi executată, în exemplul de mai sus este vorba despre subrutina cu numele **NuCumpara**. Dacă testul ia valoarea **False** atunci, instrucțiunea imediat următoare lui **Else** se execută, mai sus subrutina **Cumpara**.

**Exemplul 3 - test If pentru mai multe linii (bloc de cod)**

```
If Pret > Bani Then
      NuCumpara
End If
```

Instrucţiunea **If** pentru blocuri de cod se termină în instrucţiunea **End If**. Nu se poate scrie text pe acceaşi linie cu **If** şi după **Then**.

**Exemplul 4 - test If pentru mai multe linii (bloc de cod)**

```
If Pret > Bani Then
      MsgBox "Nu ai suficienti bani pentru tranzactie"
      NuCumpara
End If
```

În cazului instrucțiunii **If** pentru blocuri de cod nu există o limită pentru numărul liniilor care se scriu în blocul delimitat de **Then** şi **End If**. Această sintaxă se va folosi atunci cînd este nevoie de executarea mai multor instrucțiuni pentru cazul deciziei luate.

**Exemplul 5 - test If pentru mai multe linii (bloc de cod)**

```
If Pret > Bani Then
      MsgBox "Nu ai suficienti bani pentru tranzactie"
      NuCumpara
Else
      Cumpara
      MsgBox "Produsul a fost cumparat
End If
```

Pentru un **If** cu blocuri de cod clauza **Else** marchează începutul blocului care se va execută dacă rezultatul testului este **False**.

## Else If

Clauza **Else** are alternativa **Else If** care permite realizarea unor noi teste.

```
If conditie1 Tyhen
     instrucțiuni       'dacă conditie1 = True
ElseIf condiție2 Then
     instructiuni       'dacă conditie2 = True
ElseIf conditie3 Then
     instructiuni       'dacă conditie3 = True
Else
     instructiuni       'dacă toate condițiile au fost False
End If
```

## Instrucțiunea Select Case

- **Ramificare în cazul condițiilor multiple**
- **Numai 1 singură ramură se excută**
- **Cazurile inexistente se prind cu** `Case Else`

```
Select Case iLuna
     Case 9,6,4,11
          inrZile=30
     Case 1,3,5,7 To 8, 10,12
          inrZile =31
     Case 2
          inrZile =28
          If nEsteAnBisect Then inrZile=29
     Case Else
          MsgBox "Numarul lunii in afara domeniului 1-12"
Ene Select
```

# 3. Instrucțiunea Select Case

Instrucțiunea **`Select Case`** este o construcție utilă în cazul ramificărilor cu mai mult de două variante (multi-ramificări). **`Select Case`** este urmată de o **expresie** sau **variabilă** care se evaluează şi se compară apoi cu toate **`Case`**-urile care îi urmează. Testarea este secvențială, adică codul parcurge **`Case`**-urile coborînd pînă cînd ajunge la primul pentru care se găsește egalitate. Porțiunea de cod asociată respectivului **`Case`** va fi executată, apoi instrucțiunea se părăsește (deci numai un singur caz de egalitate este găsit). Dacă nu se găsește nici o egalitate, se va executa secvența de cod corespunzătoare lui **`Case Else`**, dacă aceasta lipsește nici un cod nu se execută.

Deoarece expresia de testa poate să fie o variabilă, expresie sau o constantă se poate imagina un scenariu specific butoanelor de opțiuni (**Option Button**) dintr-un cadru (**Frame**) pentru a determina opțiunea selectată de forma:

```
Select Case True
     Case optMere
          ...
     Case optPere
          ...
     Case Else
          'n-ati selectat nici un fruct
End Select
```

## Instrucțiunea For ... Next

- **Numărul de iterații este fixat**
- **Valorile de început, sfîrșit și pasul (implicit este 1) trebuie definite la intrarea în ciclu**

```
For iLuna = 1 To 12
    iZile = ZileInLuna(iLuna)
Next
```

```
iInceput=1
iSfirsit=12
For iLuna = iSfirsit To iInceput Step -1
    iZile = ZileInLuna(iLuna)
Next
```

# 4. Instrucțiunea de ciclare For ... Next

Instrucțiunea **For ... Next** este construcție standard pentru realizarea repetării unor operații (iterații) de un număr fixat de ori. Prin folosirea unei variabile pentru controlul ciclului, ce poate fi de orice tip numeric, o valoare de început, una de sfîrşit şi o valoare de increment sînt definite la intrarea în cliclu (modificări ulterioare ale acestor valori sînt ignorate dacă s-a început execuția instrucțiunii de ciclare). După fiecare execuție a instrucțiunilor din ciclu valoare de incrementare se adună la valoarea curentă a variabilei de control a ciclului iar ciclu se părăseşte daca valoare curentă este mai mare decît cea de terminare.

Valoarea de incrementare trebuie să fie negativă dacă valoare de început este mai mare decît cea de sfîrşit după cum se vede în exemplu.

Este posibil, dar neobişnuit, ca mai multe cicluri **For** să se termine în aceeaşi instrucțiune **Next** după cum se vede:

```
For i = 1 To 10
    For j = 1 To 20
        iMatrice(i,j) = 0
Next j,i
```

## Instrucțiunea Do ... Loop

- **Numărul de iterații este condiționat**
- **Folosește teste (opuse)** `While` **și** `Until`
- **Testul se face la începutul sau la sfîrșitul ciclului (în acest caz codul se execută 1 dată)**

```
iLuna = 1
Do While iLuna <=12
     iZile = ZileInLuna(iLuna)
     iLuna=iLuna+1
Loop
```

```
iLuna = 1
Do
     iZile = ZileInLuna(iLuna)
     iLuna=iLuna+1
Loop Until iLuna > 12
```

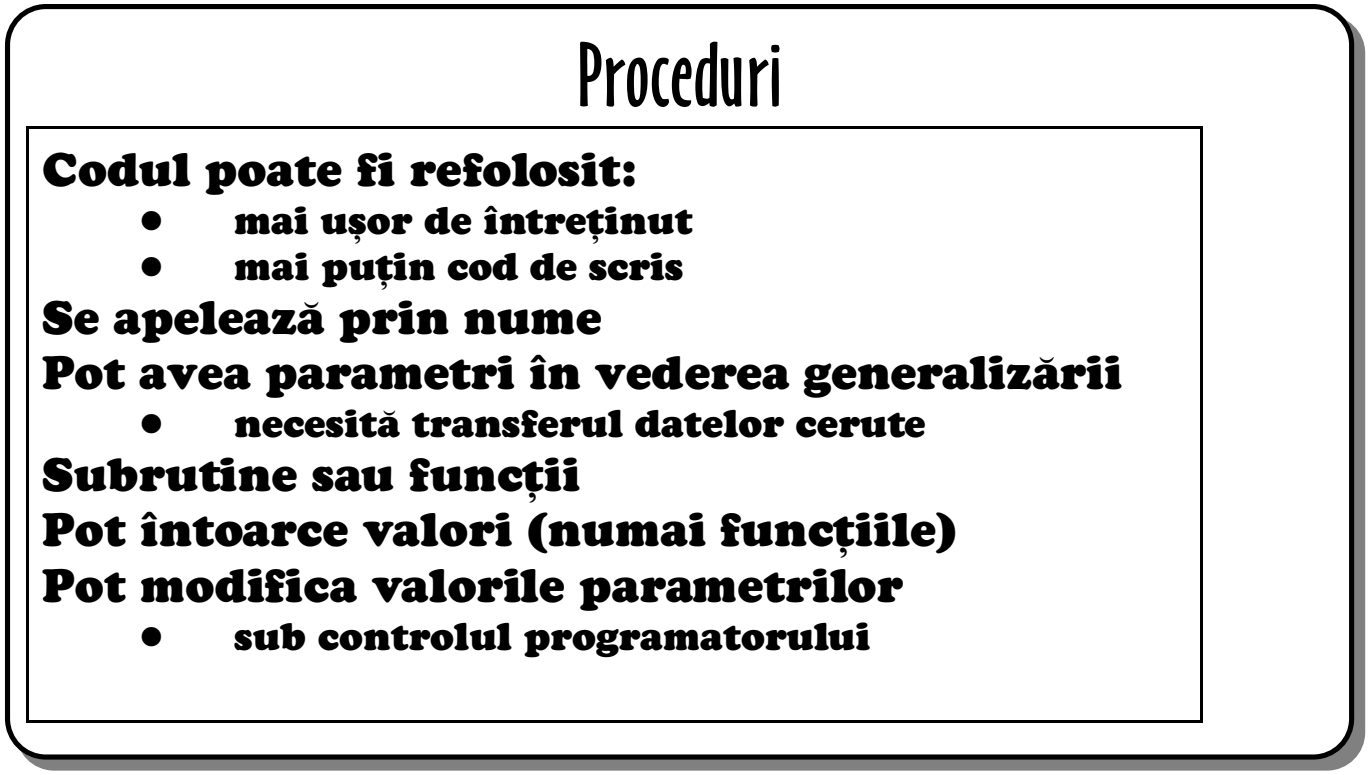
# 5. Instrucțiunea de ciclare Foor ... Loop

Construcția **Do ... Loop** permite repetarea operațiilor de un număr nedeterminat de ori (ciclu infinit). Folosind orice expresie **Boolean** în conjuncție cu **While** sau **Until** se poate controla număr de reluări ale codului. **Testul de reluare** se face în condiții normale ca şi la primul exemplu la intrarea în ciclu. Aceasta asigură ca execuția codului să înceapă numai dacă condiția de reluare este îndeplinită. **While** şi **Until** sînt exact opuse asigurînd flexibilitate în programare (**Until** este logic echivalentul lui **While Not**). Dacă testarea se face după începerea ciclului, vezi exemplul al 2-lea, codul se va executa întotdeauna de cel puțin o singură dată chiar dacă condiția de reluare nu este îndeplinită din moment ce testul se face numai după terminarea primei execuții a codului de reluat.

Este posibilă şi folosirea unei sintaxe bine de evitat din care lipsește **While** sau **Until** după cum urmează:

```
iLuna = 1
Do
     If iLuna >=12 Then Exit Do
     iZile = ZileInLuna(iLuna)
     iLuna=iLuna+1
Loop
```

Scrierea unor astfel secvențe de cod nestructurate trebuie descurajată !



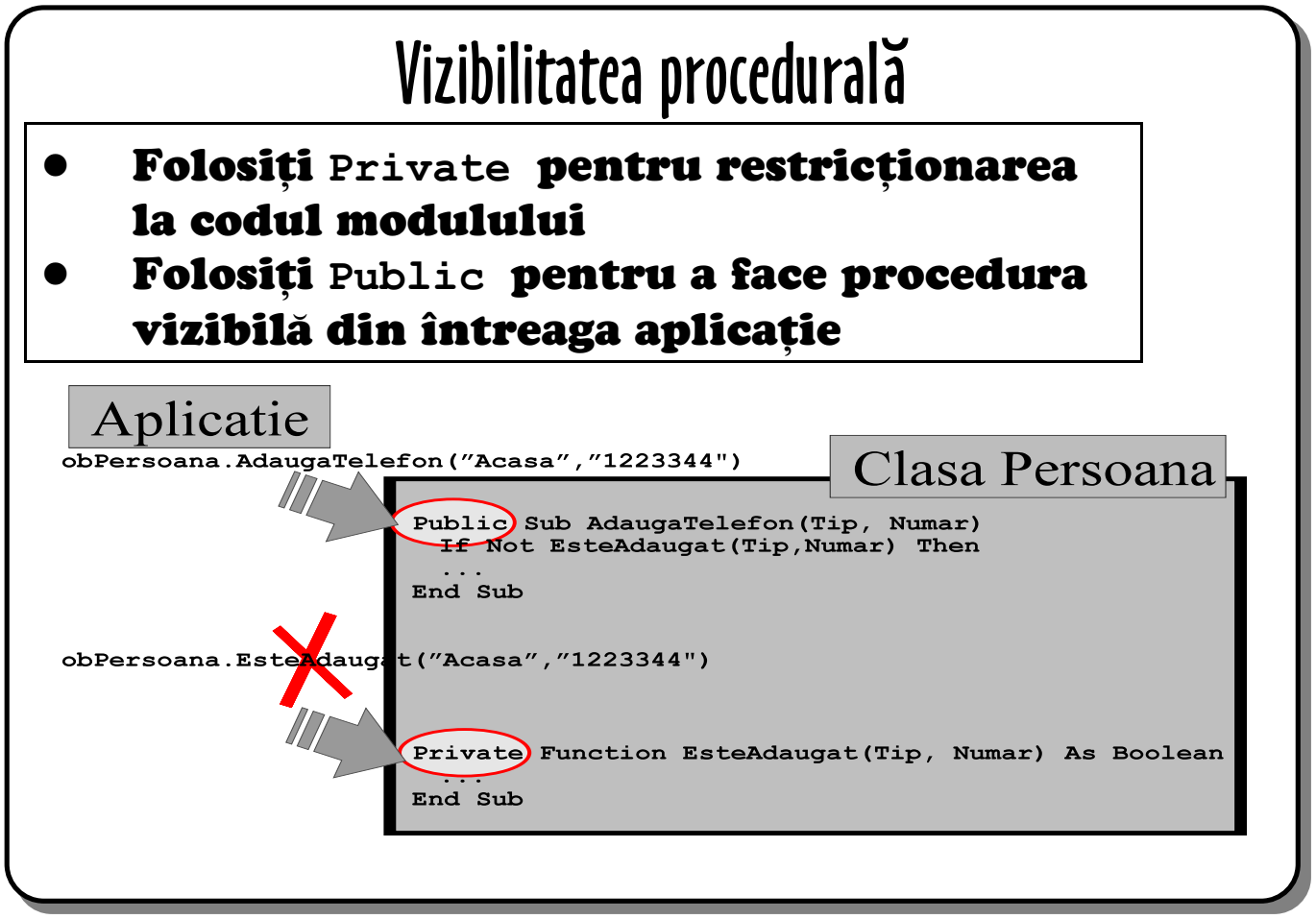

# 8. Proceduri

Procedurile permit ca o secvență de cod să fie reluată, prin apel, din mai multe porțiuni ale programului. În acest fel secvența de cod se va scrie o singură dată, iar dacă aceasta trebuie cîrpită sau extinsă aceasta se va face într-un singur loc. Aplicaţia devine din acest motiv mai mică, mai uşor de întreţinut şi se scrie mai repede.

Procedurile sînt identificate printr-un nume unic care se foloseşte la apelul lor. Cele două tipuri de proceduri sînt subrutina şi funcţia. Sintaxa lor va fi predată imediat, dar diferența esențială dintre ele este aceea că funcţia întoarce o valoare prin nume la terminare, în timp ce procedura nu.

Procedurile trebuie scrise să fie cît mai generale şi flexibile. În acest scop orice dată pe care procedura trebuie să o prelucreze va trebui transferată ca şi parametru. Astfel se realizează decuplarea procedurii de un tip particular de variabile sau obiecte din aplicație.

Sintaxa pentru apelul parametrilor dintr-o procedură poate fi ajustată pentru a opri sau permite modificarea valorilor parametrilor din interioarul procedurii. Prin folosirea lui **`ByVal`** sau **`ByRef`** pentru fiecare parametru se defineşte folosirea unei copii sau direct a valorii originale.

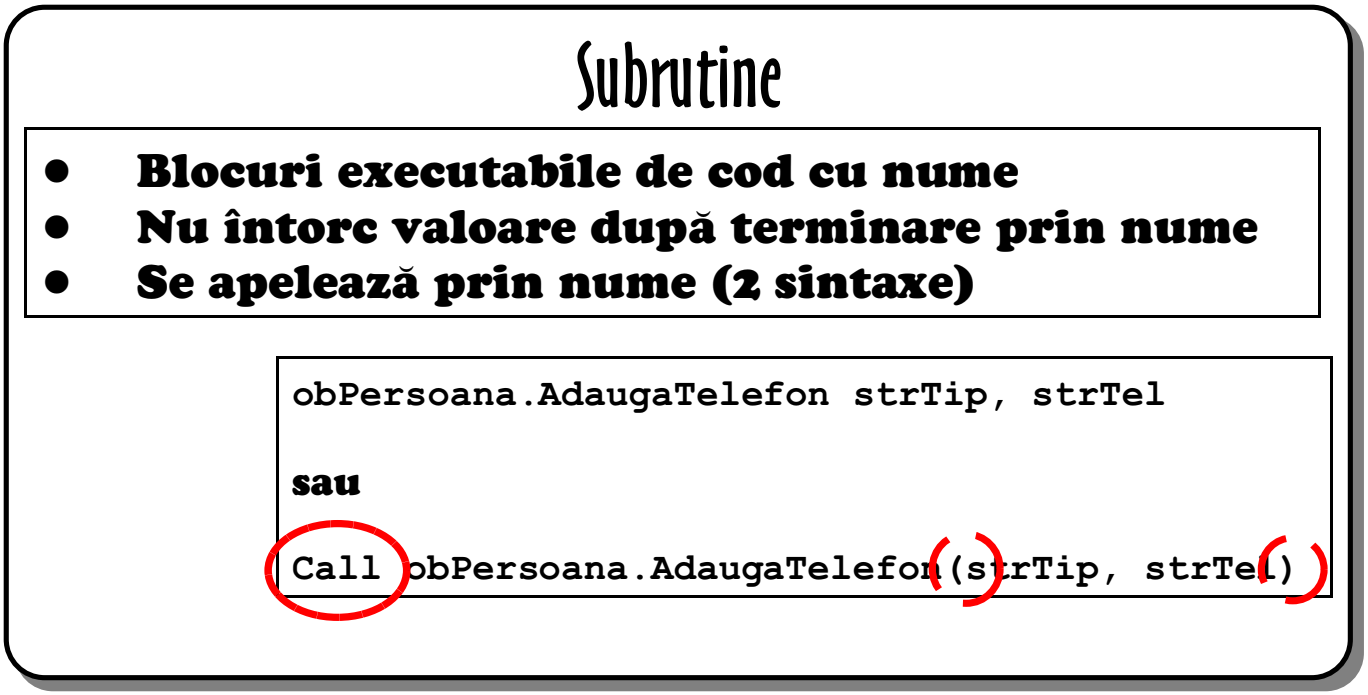



# 9. Vizibilitatea procedurilor

Procedurile, asemenea variabilelor, au şi ele vizibilitate pentru a putea controla porţiunile de cod din care vor putea fi apelate. Aceleaşi cuvinte cheie **`Public`** şi **`Private`** sînt folosite avînd acelaşi efecte.

Procedurile **`Private`** pot fi apelate numai din modulele în care s-au definit, în timp ce procedurile **`Public`** vor putea fi folosite în tot codul aplicaţiei apărînd ca şi metode în toate modulele.

În exemplu prezentat **`AdaugaTelefon`** este o procedura publică în timp ce **`EsteAdaugat`** este una privată a obiectului **`Persoana`**. **`EsteAdaugat`** este văzută în toată clasa în care s-a definit, observaţi că este apelată din **`AdaugaTelefon`**, dar nu poate fi apelată direct de un utilizator al clasei din restul aplicaţiei (prin variabila obiect **`obPersoana`**).




# 10. Subrutine

Subrutinele se declară folosind Public sau Private pentru specificarea vizibilității procedurii, urmată de Sub care indică faptul ce este o subrutină, numele subrutinei şi în final o listă de orice parametrii între parateze.

Subrutina poate fi apelată prin oricare dintre următoarele sintaxe din acelaşi modul:

```
AdaugaTelefon strTip, strTel
AdaugaTelefon "Acasa", "0263-174574"
Call AdaugaTelefon(strTip, strTel)
Call AdaugaTelefon("Acasa", "0263-174574")
```

sau din alte module respectiv obiecte prin:

```
obPersoana.AdaugaTelefon strTip, strTel
obPersoana.AdaugaTelefon "Acasa", "0263-174574"
Call obPersoana.AdaugaTelefon(strTip, strTel)
Call obPersoana.AdaugaTelefon("Acasa", "0263-174574")
```

Valorile sau variabilele apelului de subrutină sînt asociate variabilelor cu nume din lista parametrilor subrutinei, astfel la nivelul subrutinei datele vor fi referite întotdeauna prin aceleaşi nume.

# Subrutine pentru tratarea evenimentelor

- **Definite de VB**
  **- numele și argumentele nu se pot modifica**
- **Se poate modifica vizibilitatea**
  **- implicit este** `Private`**, dar poate fi făcută** `Public`
- **Pot fi apelate din cod;**
  **- de ex.** `ButtonClick` **apelează** `Click` **de meniu.**

```
Private Sub cmdOK_Click()
     ...
End Sub
```

```
Public Sub cmdOK_Click()
     ...
End Sub
```

## 10.1 Subrutine pentru tratarea evenimentelor

Rutinele de tratatre a evenimentelor sînt subrutine apelate de runtime-ul VB ca răspuns la acțiunile detectate de sistemul de operare. Nu se pot modifica numele sau detaliile legate de parametrii de apel.

Se poate modifica însă vizibilitatea rutinei la **`Public`** sau **`Private`**, iar rutinele pot fi apelate asemenea oricărei proceduri. Este o cerință normală ca unele secvențe de cod asociate unor butoane de comandă să poată fi apelate din anumite secvențe de program, de exemplu în momentul în care se face **`Click`** pe un articol de meniu al unui toolbar.

## Functii

- **Întorc o valoare prin numele lor la terminare**
- **Pot fi tratate ca subrutine**

```
bStare = obPersoana.Adauga("Acasa","0263-174574")
sau
obPersoana.Adauga("Acasa","0263-174574") 'se presupune
                                         'ca totul e OK
```

```
Public Function Adauga(strTip As string, strNr As String) _
                                                    As Boolean
     Adauga = False
     If AdaugaPersoana(strTip, strNr) Then
          Adauga = True
     End If
End Function
```

# 11. Funcţii

Funcţiile pot face tot ceea ce fac subrutinele, dar după terminarea lor întorc un rezultat codului din care au fost apelate (codul apelant). Dacă luăm exemplul funcţiilor matematice Sin, Cos, Tan etc. toate acestea primesc valori de intrare, realizează calcule şi întorc un răspuns. Funcţiile VB sînt mult mai flexibile, pot primi 0 sau mai mulţi parametrii şi întorc ca valoare obiecte sau tablouri, dar şi orice tip de dată simplu.

Funcţiile se declară folosind **`Public`** sau **`Private`** pentru vizibilitate, urmează apoi cuvîntul Function care indică faptul ca urmează o funcţie, numele funcţiei, lista parametrilor, cuprinsă între paranteze. Spre deosebire de subrutine, doarece funcţiile întorc o valoare, tipul de dată al valorii întoarese trebuie declarat la sfîrşitul isntrucţiunii. Funcţia se termină cu instrucţiunea **`End Function`**.

Valoarea întoarsă se defineşte atribuind numelui funcţiei valoarea dorită după cum se observă în exemplu. Funcţia **`Adauga`** apelează o altă funcţie numită **`AdaugaPersoana`** care întoarce un rezultat **`Boolean`**. Observaţi că, iniţial, prin atribuirea **`Adauga = False`** funcţia ia valoarea **`False`** însă în cazul funcţiilor **`Boolean`** ea nu este necesară deoarece **`False`** este valoarea lor implicită (dar este bine de scris deoarece face codul mai clar).

Pentru funcţiile care întorc obiecte instrucţiunea **`Set`** trebuie folosită la atribuire.

Sintaxa pentru apelul din același modul este:

```
bStare = Adauga(strTip, strNr)
bStare = Adauga("Acasa","0263-174574")
```

Pentru apelul din afara modulului se scrie:

```
bStare = obPersoana.Adauga(strTip,strNr)
bStare = obPersoana.Adauga("Acasa","0263-174574")
```

Este posibil ca funcția să fie tratată asemenea unei subrutine, caz în care valoarea întoarsă se ignoră. În acest caz sintaxele de apel devin:

Apel din același modul:

```
Adauga strTip, strNr
Adauga "Acasa","0263-174574"
Call Adauga(strTip, strNr)
Call Adauga("Acasa","0263-174574")
```

Apel dintr-un alt modul sau prin obiect:

```
obPersoana.Adauga strTip, strNr
obPersoana.Adauga "Acasa","0263-174574"
Call obPersoana.Adauga(strTip, strNr)
Call obPersoana.Adauga("Acasa","0263-174574")
```

## Mecanismul de transfer al parametrilor

- **2 mecanisme de transfer**
- **Folosiți `ByVal` pentru a crea și lucra cu o copie a datelor**
- **Folosiți `ByRef` pentru a lucra direct cu dalele (implicit)**

```
Public Sub TransferBani(ByRef Suma As Currency, ByRef _
inCont As String)
    ...
End Sub
```

```
Public Sub TransferBani(ByVal Suma As Currency, ByVal _
inCont As String)
    ...
End Sub
```

# 12. Mecanismul de transfer al parametrilor

**`ByVal`** se foloseşte în fața unei definiții de parametru pentru a-i spune lui VB să creeze o copie a datei de transferat şi să lucreze cu respectiva copie în locul originalului. Dacă se fac modificări ale datei în interiorul procedurii, acestea nu vor avea efect asupra datelor originale din afara procedurii.

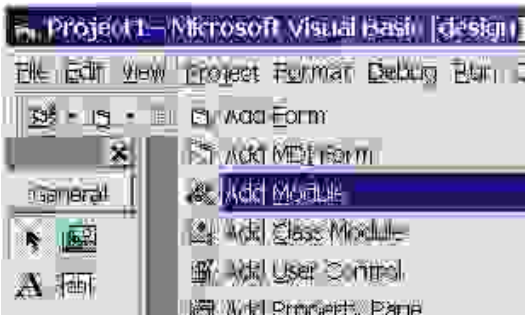
**`ByRef`** se folseşte pentru a-i spune lui VB să lucreze cu o referință la datele originale din memorie (nu se face o copie a lor). Dacă se fac modificări ale acestor date în interiorul procedurii acestea afectează datele originale din afar ei. Metoda de transfer este mai rapidă dar şi mai periculoasă, din nefericire este şi comportamentul implicit a lui VB dacă **`ByVal`** sau **`ByRef`** nu sînt specificate.

**Tip:**

Este bine ca **`ByVal`** sau **`ByRef`** să fie explicit scrise la definirea procedurilor permițînd astfel ca alții să fie siguri asupra alegerii făcute de noi.

# Module de cod

- **Folosite pentru rutine utilitare**
- **Rutinele `Public` sînt disponibile în toată aplicația**
- **Același modul poate fi partajat între mai multe aplicații**




# 13. Module de cod

După ce procedurile aplicației au fost rescrise în așa fel încît să nu se mai refere la variabile sau obiecte ale unui formular specific nu mai este necesar ca respectivele rutine să fie asociate unui formular. Devenind rutine cu scop general sau rutine utilitare vor putea fi salvate în module de cod standard.

Conform regulilor de vizibilitate procedurile **`Public`** sînt vizibile în toată aplicația, iar cele **`Private`** numai în modulul în care s-au definit.

Modulele au extensia **.bas**. Dacă manifestați grijă cu privire la modul de grupare a procedurilor în modul puteți construi biblioteci de cod care pot fi incluse în alte aplicații.

**Tip:**

Grupați într-un modul de cod rutine similare pentru a simplifica gestionarea codului de bibliotecă partajat. De exemplu, apelurile de funcții API se grupează într-un modul, funcțiile matematice într-un altul etc.