**SUBPROGRAME**

**Proceduri si functii**

O aplicatie Visual FoxPro poate fi proiectata modular, iar modulele pot fi programe cu extensia ***.prg*** (subprograme), proceduri si functii. programele cu extensia ***prg*** pot fi situate toate in acelasi director sau in directoare diferite, la fel ca si tabelele de date. Daca se gasesc in diferite se poate utiliza setarea:

*SET PATH TO <lista\_directoare>*

**Apelul programelor (subprogramelor)** cu extensia ***.prg*** se face cu comanda :

*DO <nume\_fisier\_program>*

iar numele fisierului poate sa nu contina extensia. In situatia cand s-a generat un fisier program cu generatorul de ecrane, cu generatorul de meniuri sau utilitarul ***RQBY***, extensia fisierului program trebuie sa figureze obligatoriu.

Procedurile si functiile se pot gasi in programul apelant, la sfarsit sau intr-un fisier separat numit ***fisier de proceduri.*** Executarea unui subprogram a unei proceduri sau a unei functii se incheie la intalnirea uneia din comenzile: ***RETURN, CANCEL, QUIT***  sau la intalnirea sfarsitului de fisier. Comanda ***RETURN*** are sintaxa urmatoare:

*RETURN [<expr> I TO MASTER I TO <nume\_program>]*

**Definitie: Vom numi modul un subprogram, o procedura sau o functie.**

Un modul poate la randul lui sa apeleze unul sau mai multe module, apelul facandu-se cu comanda ***DO*** pentru subprograme si cu numele functiei pentru functii. Se pot utiliza si comenzi ***DO*** imbricate (subprogramul apelat sa apeleze la randul sau alt subprogram) acceptandu-se maxim 32 nivele de imbricare (la varianta ***DOS***).

Daca instructiunea ***RETURN*** figureaza fara clauze, dupa executarea modulului apelat se revine in programul apelant la instructiunea urmatoare apelului. Daca exista clauza <*expr*>, aceasta reprezinta valoarea pe care o transmite modulul programului apelant. Clauza ***TO MASTER*** face ca programul principal sa preia controlul dupa incheierea executarii modulului apelat, iar ***TO <nume\_program>*** preda controlul subprogramului cu numele specificat.

Comanda ***CANCEL*** intrerupe executarea programului si preda controlul ferestrei de comenzii ***QUIT,*** forteaza incheierea executarii unui program si face iesirea din mediul ***FOX*** .

Un subprogram mai poate fi si suspendat temporar, suspendarea se face folosind comanda:

***SUSPEND***

iar reluarea executarii se face cu comanda:

***RESUME***

In afara subprogramelor definite de utilizator care au extensia ***.prg*** in ***Visual FoxPRo*** se utilizeaza proceduri si functii sau pe scurt ***UDF (“User Defined Function”-*** functii definite de utilizator***).*** Ele se pot situa:

    la sfarsitul fisierului sursa ce contine programul care le apeleaza si in acest caz pot fi folosite numai in acest program;

    in fisiere speciale de proceduri si functii.

**Declararea si apelarea functiilor si procedurilor**

Declararea functiilor

*FUNCTI <nume\_functie>*

 *PARAMETERS <lista\_parametri>*

 *<instructiuni>*

 *………………………………………..*

 *RETURN <valoare>*

**Apelul functiilor**

 Apelul functiilor se face precizand numele functiei si lista de parametri, daca functia are parametri:

 *<nume\_functie>( [<lista\_parametri>])*

 Parametri utilizati la apel trebuie sa respecte numarul si ordinea parametrilor descrisi cu:

 *PARAMETERS.*

**Declararea procedurilor**

*PROCEDURE <nume\_procedura>*

 *PARAMETERS <lista\_parametri>*

 *<instructiuni>*

 *………………………………………….*

 *RETURN <valoare>*

**Exemplu:**

*set talk off*

 *clear*

 *&& Se calculeaza TVA intr-o tabela cu produse*

 *&& cu structura COD, CANTITATE, PRET, VALOARE, TVA*

 *accpt ‘Numele tabelei:’ to numetab*

 *do proc\_TVA with numetab*

 *return*

 *procedure proc\_TVA*

 *parameters s*

 *use &s*

 *replace all tva with valoare\*0.19*

 *use*

 *return*

**Apelul procedurilor**

 Pentru a apela o procedura se utilizeaza comanda ***DO***, urmata de numele procedurii si lista parametrilor:

 *DO <nume\_procedura> [WITH <lista\_parametri>].*

 La fel ca la functii parametrii utilizati la apel trebuie sa respecte numarul si ordinea parametrilor descrisi cu *PARAMETERS.*

 **Observatii: Mediul Fox accepta apelarea recursiva a subprogramelor; numarul de reapelari este insa mic.**

**Fisiere pentru functii si proceduri**

Este avantajos ca toate functiile si procedurile sa fie grupate intr-un fisier, deoarece se gasesc usor cand se fac modificari in ele si pot fi utilizate in mai multe aplicatii prin precizarea fisierului care le contine. Deschiderea inchiderea unui fisier de subprograme se face cu comanda:

 *SET PROCEDURE TO [<nume\_fisier>]*

Daca ***<nume\_fisier>*** este prezent, se deschide fisierul cu numele specificat ce contine subprograme; altfel se inchide: Inchiderea se mai poate face si cu comanda:

 *CLOSE PROCEDURE*

 Fisierul de subprograme se pozitioneaza de obicei in directorul in care se afla programul principal al aplicatiei care utilizeaza fisierele respective.

 **Exemplu:**

*Proc\_2.prg*

 *set talk off*

 *clear*

 *set procedure to proc.prg*

 *accept ‘Introduceti tabela:’ to s*

 *input ‘Nr. de inregistrare:’ to n*

 *? TVA(s,n)*

 *do p\_1 with s,n*

 *close procedure*

 *return*

 *Proc.prg*

 *function TVA*

 *parameters s,n*

 *use &s*

 *locate for recno()=n*

 *vtva=tva*

 *use*

 *return vtva*

 *procedure p\_1*

 *parameters s,n*

 *use &s*

 *edit record n noedit*

 *delete*

 *pack*

 *use*

 *return*

**Parametrii si variabilele locale ale functiilor si procedurilor**

 Lucrul cu module de programe impune rezolvarea unor probleme referitoare la variabilele programului apelant si a modulului. Ne intereseaza domeniul de “vizibilitate” a variabilelor din programul apelant si modul si ce se intampla daca in program si modul exista variabile cu acelasi nume. Vom trata in continuare aceste aspecte.

Consideram un program principal care apeleaza mai multe module. Orice variabila folosita in programul principal este recunoscuta in module si poate fi modificata. In programul principal variabila va acea valoarea modificata in cadrul modulelor. Acelasi lucru se intampla si daca modulele se apeleaza imbricate fiind mai multe nivele de subprograme.

 **Exemplu:**

 *set talk off*

 *clear*

 *val1=5*

 *do p1*

 *? val1*

 *do p2*

|  |
| --- |
|  |

 *? val1*

 *return*

 *procedure p1*

 *val1=val1+3*

 *return*

 *procedure p2*

 *val1=val1+5*

 *return*

 *set talk off*

 *clear*

 *val1=5*

 *do p1*

 *? val1 –Se va afisa 13*

 *return*

 *procedure p1*

 *val1=val1+3*

 *do p2*

 *return*

 *procedure p2*

 *val1=val1+5*

 *return*

Am vazut ca variabile din programul principal sunt recunoscute in toate modulele apelate din program. Situatia reciproca nu are loc: variabilele unui modul nu sunt recunoscute decat in modulele apelate de acesta.

 **Exemplu:**

*set talk off*

 *clear*

 *do p1*

 *&& se afiseaza 8*

 *? val1*

 *&& eroare*

 *return*

 *procedure p1*

 *val1=3*

 *do p2*

 *? val1*

 *return*

 *procedure p2*

 *val1=val1+5*

 *return*

Modificarea domeniului de “vizibilitate” a variabilelor modulelor se face prin definirea lor ca variabile ***publice*** (globale) sau ***private*** (locale).

 Variabilele publice se definesc cu comanda:

 *PUBLIC <lista\_variabile>*

 *Public [ARRAY] <masiv>(expN1>,<expN2>)*

In urma comenzii se creeaza variabilele sau masivele specificate si se initializeaza cu *.F.* Eroarea din exemplul anterior poate fi inlaturata prin definirea lui ***val1***ca variabila de tip ***PUBLIC.***

 *set talk off*

 *clear*

 *public val1*

 *do p1*

 *? val1*

 *return*

Variabilele private (locale) sunt recunoscute in programul curent si in modulele apelate de acesta. Se declara cu comanda:

 *PRIVATE <lista\_variabile>*

 *PRIVATE ALL [ LIKE <sablon>/ EXCEPT <sablon>]*

Spre deosebire de comanda ***PUBLIC*** care creeaza variabilele globale, comanda ***PRIVATE***  le declara numai.

 **Exemplu:**

 *set talk off*

 *val1=5*

 *val2=8*

 *do p1*

 *? val1, val2*

 *&& se afiseaza 5 80 deoarece val1 este private*

 *return*

 *procedure p1*

 *private val1*

 *val1=50*

 *val2=80*

 *? ‘ Val1 Val2’*

 *? val1, val2*

 *return*

 In situatia cand se doreste utilizarea variabilelor cu acelasi nume in diverse sectiuni ale programului, variabilele pot fi declarate *“regionale”* : sunt cunoscute numai in sectiunea specificat[ de programator. In program se va marca inceputul sectiunii cu ajutorul directivei de compilare:

 *# REGIONAL <numar>* iar variabilele vor fi precizate ca locale, folosind instructiunea:

 *REGIONAL <lista\_variabile>*

O variabila ***<lista\_variabile>*** este utilizabila pana la intalnirea unei noi declaratii ***REGIONAL*** care sa o includa, sau pana la sfarsitul programului. Sectiunea de localizare a variabilelor se incheie la intalnirea unei noi directive ***# REGIONAL***  sau a sfarsitului de fisier. Aceleasi variabile pot fi declarate in zone diferite ale programului, fara a avea legatura intre ele.

 **Exemplu:**

*set talk off*

 *clear*

 *# region 1*

 *regional a, b, c*

 *a=5*

 *b=7*

 *c=8*

 *# region 2*

 *regional a, b*

 *? a – Eroare ! Variabila a nu este gasita.*

 *a=9*

 *b=10*

 *? a, b, c –* Se afiseaza 9 10 8. Variabila ***c*** este recunoscuta deoarece nu a fost trecuta in lista variabilelor din zona 2.

 return

 **Transmiterea parametrilor modulelor de program**

In cazul proiectarii unei aplicatii mari se impune utilizarea modulelor de program, mai ales cand programarea nu se face de o singura persoana. Comunicarea modulelor cu un program apelant sau cu alte module se face prin intermediul parametrilor asociati modulelor.

 Ca si in alte limbaje si in ***Visual FoxPro*** avem doua modalitati de transmitere a parametrilor:

    ***prin valoare***

    ***prin referinta.***

 La transmiterea prin ***valoare,*** o modificare a parametrului in modul nu afecteaza valoarea avuta de acelasi parametru in programul apelant. Daca parametrul va fi transmis prin ***referinta,*** in urma executarii modulului, valoarea sa din programul apelant va ramane cea din modul. Mecanismul de transmitere al parametrilor este urmatorul:

    variabilele care intra in modul pentru prelucrare sunt preluate de parametric la apelarea subprogramului, respectandu-se ordinea si tipul parametrilor descrisi in comanda ***PARAMETERS <lista\_parametri>;***

    daca parametrii modulului au fost transmisi prin referinta, la incheierea executarii modulului continutul parametrilor este preluat de variabilele corespunzatoare ale programului apelant.

 Lista parametrilor unui modul se declara (dupa cum am vazut) cu comanda:

 *PARAMETERS <lista\_parametri>,*

care este prima comanda a modulului. Pentru a afla numarul de parametri transmisi modulului curent, se poate folosi functia:

 *PARAMETERS ()*

In urma precizarii modului de transmitere a parametrilor (prin valoare sau referinta) se foloseste comanda:

 *SET UDFPARMS TO VALUE / REFERENCE*

 *SET UDFPARMS TO VALUE* transmite parametrii prin valoare, iar *SET UDFPARMS TO REFERENCE*  transmite parametrii prin referinta.

**Exemplu:**

set talk off

 clear

 set udfparms to value

 a=1

 =fun\_1(a)

 ? a

 && 1

 set udfparms to reference

 a=5

 =fun\_1(a)

 ? a

 && se afiseaza 7

 return

 function fun?1

 parameters s

 a=a+2

 return a

**Observatie: Masivele se transmit ca parametri prin referinta.**

**Problema rezolvata:** Sa se scrie o functie care sa interclaseze doua masive ordonate crescator.

**Rezolvare:**

 set talk off

 clear

 dimension a(100), b(100), c(100)

 input ‘Nr. elemente a: ‘ to n

 input ‘Nr. elemente b: ‘ to m

 ? ‘Intr. elem. lui a’

 for i=1 to n

 input ‘a:’ to a(i)

 endfor

 ? ‘Intr. elem. lui b’

 for i=1 to m

 input ‘b:’ to b(i)

 endfor

 set udfparm to reference

 =intercl(a,b)

 for i=1 to n+m

 ? c(i)

 endfor

 return

 function intercl

 parameters a, b

 i=1

 j=1

 k=0

 do while (i<=n) .and. (j<=m)

 k=k+1

 if a(i)<b(j)

 c(k)=a(i)

 i=i+1

 else

 c(k)=b(j)

 j=j+1

 endif

 enddo

 if i<n

 for j=I to n

 k=k+1

 c(k)=a(j)

 endfor

 else

 for i=j to m

 k=k+1

 c(k)=b(i)

 endfor

 endif

 return c

***Memento***

***PROCEDURI***

Structural o procedură, cuprinde următoarele elemente:

**PROCEDURE *nume\_procedură*** -specifică numele procedurii create

**PARAMETERS *listă\_parametri*** -declararea parametrilor formali d eintrare

***Bloc de comenzi*** -bloc d e comenzi,corpul procedurii

**RETURN [*expresie*]** -ieşirea dinprocedură, opţional cu returnarea unei expresii calculate.

* PROCEDURE – este o declaraţie în interiorul unui program, care specifică începutul uneiproceduri şi defineşte numele procedurii. Numele procedurii poate începe cu o literă sau caracterul “\_” (underscore) şi poate conţine orice combinaţie de litere, numere şi caractereunderscore. Lungimea maximă a numelui este de 254 caractere;
* PARAMETERS – asigură trecerea parametrilor din programul apelant în procedură. Se pottransmite maxim 27 de parametri în procedură. Lista parametrilor poate cuprinde: expresii,variabile de memorie, nume de tabele, nume de câmpuri din tabele de date, constante, caresunt transmise prin referinţă;
* RETURN [expresie] – returnează controlul şi opţional o expresie, în programul apelant;
* Parametrii pot fi transmişi în procedură, prin includerea clauzei PARAMETERS înprocedură, sau prin plasarea unei liste de parametri imediat după PROCEDUREnume\_procedură. Lista se inchide între paranteze, iar parametrii sunt separaţi cu virgulă.

**DO <nume\_procedură> | <nume\_program> WITH <listă\_parametri>**- Apelul unei proceduri sau program

**Funcţii**

Elementele unei funcţii definite de utilizator (UDF) sunt:

**FUNCTION nume\_funcţie** -specifică numele funcţieicreate

***bloc de comenzi*** -bloc de comenzi, corpul funcţie

**RETURN [expresie]** -ieşirea din funcţie, opţional cu returnarea unei expresii calculate

* Implicit parametrii sunt transmişi în funcţie prin valoare. Numărul maxim de parametri care poate fi transmis, este 27.
* Parametrii pot fi transmişi în funcţie, prin includerea clauzei PARAMETERS în funcţie, sau prin plasarea unei liste de parametri imediat după FUNCTION nume\_funcţie. Lista se inchide între paranteze, parametrii sunt separaţi cu virgulă.

**Funcţii predefinite în Visual Fox Pro**

**SUM** – calculează suma.

**AVERAGE** – calculează media.

**TOTAL**- întoarce numărul de înregistrări.

**COUNT** – calculează numărul de înregistrări.

**FIELD()** – întoarce numele cîmpului curent.

**EOF**(<zona>) - testează sfîrşitul de fişier din zona <zona>.

**BOF**(<zona>) - testează începutul fişierului.

**RECNO**(<zona>) - întoarce numărul înregistrării curente.

**RECCOUNT** (<zona>) - întoarcenumărul de înregistrări.

**MONTH()** determină luna dintr-o dată calendaristică,

**DAY()** – determină ziua,

**YEAR()** – determină anul.

**Situaţii didactice de învăţare**

1. Activaţi tabelul *Student*.
2. Afişaţi suma tuturor cîmpurilor numerice din tabelul *Student*.
3. Afişaţi suma burselor tuturor studenţilor.
4. Memorizaţi suma burselor într-o variabilă *s*, iar apoi afişaţi valoarea ei pe ecran.
5. Calculaţi suma burselor tuturor fetelor din grupă.
6. Calculaţi media bursei grupei.
7. Calculaţi media burselor studenţilor din Chişinău.
8. Afişaţi cîte înregistrări aveţi în tabelul *Student* (n records).
9. Calculaţi cîte înregistrări aveţi în tabelul *Student,* astfel încît pe ecranul principal să apară doar numărul de înregistrări.
10. Afişaţi numele cîmpului 2.
11. Creaţi un fişier de comandă *fc1*, în care veţi:
* Declara variabila *a* de tip ***numeric*** cu valoarea *100* şi afişa valoarea ei pe ecran;
* Declara variabila *b* de tip ***logical*** cu valoarea *adevărat* şi afişa valoarea ei pe ecran;
* Seta formatul de afişare a datelor calendaristice de tip German;
* Declara variabila *c* de tip ***date*** cu valoarea *20.06.1986* şi afişa valoarea ei pe ecran;
* Declara variabila *d* de tip ***character*** cu valoarea *Gutu Ion* şi afişa valoarea ei pe ecran;

Salvaţi şi executaţi fişierul *fc1*.

1. Afişaţi toate fişierele de comandă.
2. Scrieţi un miniprogram *fc2* care afişează data de astăzi. Executaţi fişierul *fc2.*
3. Scrieţi un miniprogram *fc3* în care veţi utiliza ciclul SCAN pentru a determina dacă în cîmpul*date\_adaugatoare* de tip *memo* a fiecărui student este înscris ceva şi dacă da - veţi afişa numele şi prenumele studentului şi conţinutul cîmpului memo, iar dacă nu – numele şi prenumele studentului şi mesajul: „No notes!!!”.

*Indicaţii:*Utilizaţi IIF.

1. Scrieţi un program ce determină dacă ziua de naştere a studentului a fost sau nu încă şi afişează mesajele corespunzătoare.
2. Scrieţi un miniprogram care aşteaptă introducerea numerelor de la tastieră şi le însumează pînă suma lor nu va fi mai mare de 200.

*Indicaţii:*Utilizaţi comanda INPUT [<expresie>] TO <variabila> pentru a introduce date de la tastieră şi a le memoriza în variabile.

1. Descrieţi şi apelaţi o procedură *Fete*, care va căuta (cu ajutorul LOCATE) toate fetele din *BD Student* şi va afişa numărul total al acestora.
2. Descrieţi şi apelaţi o procedură recursivă care va aştepta introducerea de la tastieră a unui număr din intervalul [0;100]. (Dacă utilizatorul introduce un număr din afara intervalului – procedura se apelează recursiv şi utilizatorului i se cere din nou introducerea numărului; dacă utilizatorul introduce un număr din interval – apare mesajul: „*Bravo, numărul ... este din intervalul [0;100]!”)*

*Indicaţii: Utilizaţi BETWEEN(expresie,valoare1,valoare2), care întoarce True dacă valoare1<=expresie<=valoare2.*

1. Scrieţi un miniprogram care calculează cîţivip studenţi aveţi în grupă, conform BD Student (vip studenţii sunt consideraţi studenţii la care bursa>400 lei):
2. Cu ajutorul ciclului scan;
3. Cu ajutorul ciclului while.