Programarea si utilizarea calculatoarelor - MATLAB

Manipularea datelor prin fişiere de intrare - ieşire

Autori: Calin Vaida, Bogdan Gherman, Doina Pisla

Cluj-Napoca 2014

Cuprins

- Funcţii pentru manipularea datelor
  - Funcţia – fopen
  - Funcţia – ferror
  - Funcţia – fgetl
  - Funcţia – frewind
  - Funcţia – ftell
  - Funcţia – fwrite
  - Funcţia – fprintf
  - Funcţia – fclose
  - Funcţia – feof
  - Funcţia – fgets
  - Funcţia – fseek
  - Funcţia – fread
  - Funcţia – fscanf
- Iteraţia cu număr necunoscut de paşi cu test final

Funcţii pentru manipularea datelor

Funcţia - **fopen**

Deschide/creează un fişier nou sau furnizează informaţii despre fişiere deja deschise. Funcţia are următoarele sintaxe, unde o parte din parametrii sunt opţionali.

fileID = fopen(nume_fis)

Deschide fişierul cu numele **nume_fis** cu acces pentru citirea datelor şi returnează (ca valoare pentru **fileID**) un număr întreg (integer).

fileID = fopen(nume_fis, permission)

Dacă se foloseşte şi opţiunea **permission**, aceasta permite configurarea modului de acces la fişierul **nume_fis**.

fileID = fopen(nume_fis, permission, machineformat)

Dacă se foloseşte şi opţiunea **machineformat**, aceasta permite configurarea modului de citire şi scriere a biţilor şi bytes-lor în fişierul **nume_fis,** pe baza unor standarde.

Funcţia - **fopen**

fileID = fopen(nume_fis, permission, machineformat, encoding)

Dacă se foloseşte şi opţiunea **encoding**, aceasta permite configurarea modului de codificare a caracterelor din fişierul **nume_fis**.

[fileID, message] = fopen(nume_fis, ...)

Dacă se foloseşte şi opţiunea **message**, comanda va returna un mesaj de eroare specific dacă operaţiunea de deschidere a fişierului **nume_fis** nu se poate realiza. În caz contrar, mesajul va fi un şir de caractere gol.

fIDs = fopen('all')

Comanda va genera un vector linie cu toate identificatoarele (valorile întregi ale fileID) tuturor fişierelor deschise.

[nume_fis, permission, machineformat, encoding] = fopen(fileID)

Funcţia - **fopen**

Comanda de mai sus va returna toate proprietăţile utilizate la apelarea funcţiei **fopen** pentru fişierul cu identificatorul **fileID**. Dacă identificatorul nu există, toate argumentele vor fi generate ca şiruri de caractere goale.

Se vor prezenta în continuare valorile care pot fi setate cai şi parametri opţionali în cazul deschiderii unui fişier.
Parametrul **permission** acceptă următoarele valori care se referă la modul de acces al datelor:
'r' – citire (parametrul nativ);
'w' – scriere; dacă fişierul este unul existent, această opţiune îi va şterge conţinutul;
'a' – scriere prin adăugarea datelor la sfârşitul conţinutului unui fişier existent;
'r+' – citire şi scriere;
'w+' – citire şi scriere cu ştergerea conţinutului existent;
'a+' – citire şi scriere cu adăugarea noului conţinut la sfârşitul fişierului.

Opţiunea **machineformat** permite definirea modului de citire a datelor în formă binară (de la stânga la dreapta sau de la dreapta la stânga).

Funcţia - **fclose**

Închide unul sau toate fişierele deschise.
Sintazele posibile pentru această funcţie sunt:
fclose(fileID)

Închide fişierul cu identificatorul **fileID.**

fclose('all')

Închide toate fişierele deschise.

status = fclose(...)

Returnează 0 (zero) dacă operaţia de închidere a fost reuşită. În caz contrar, **status** va primi valoarea -1.

Funcţia - **ferror**

Funcţia **furnizează informaţii** despre mesajele de eroare la manipularea fişierelor de intrare /ieşire (I/O).

Funcţia - **feof**

Funcţia **testează** dacă s-a atins poziţia de sfârşit a fişierului (end-of-file).
Sintaxa este:
status=feof(fileID)

Dacă o funcţie anterioară a setat poziţia curentă pe capătul fişierului valoarea returnată (**status**) va fi 1, în caz contrar va fi 0.

Funcţia - **fgetl**

Citeşte o linie dintr-un fişier fără a include şi caracterul de linie noua (\n).
Sintaxa este:
tline = fgetl(fileID)

Astfel **tline** va fi un şir de caractere cu excepţia cazului în care linia conţine doar caracterul de sfârşit de fişier (eof) caz în care **tline** va avea valoarea numerică **-1**.

Funcţia - **fgets**

Citeşte o linie dintr-un fişier şi include şi caracterul de linie noua (\n).
Sintaxa este:
tline = fgets(fileID)

Astfel **tline** va fi un şir de caractere cu excepţia cazului în care linia conţine doar caracterul de sfârşit de fişier (eof) caz în care **tline** va avea valoarea numerică **-1**.
Faţă de funcţia **fgetl** funcţia **fgets** are o sintaxă suplimentară:

tline = fgets(fileID, nchar)

Această sintaxă permite citirea a cel mult **nchar** caractere dintr-o linie. Dacă linia are mai puţine caractere, nu se vor citi caractere suplimentare după întâlnirea caraterelor speciale de linie nouă şi/sau sfârşit de fişier.

Funcţia - **frewind**

Poziţionează cursorul la începutul fişierului.
Sintaxa este:
frewind(fileID)

Funcţia - **fseek**

Poziţionează cursorul la o poziţie specificată în fişier.
Sintaxele funcţiei sunt: *fseek(fileID, offset, origin)*

În această formă, funcţia va seta poziţia curentă în fişierul cu identificatorul **fileID** la un număr de **offset** bytes faţă de poziţia definită de **origin**.
Astfel, **offset** poate lua valori pozitive, negative şi valoarea zero. Parametrul **origin** are trei valori posibile:

'bof' sau -1 - Începutul fişierului;
'cof' sau 0 - Poziţia curentă;
'eof' sau 1 – Sfârşitul fişierului.

status = fseek(fileID, offset, origin)

Această sintaxă va returna 0 (zero) dacă operaţia este reuşită şi -1 în caz contrar.

Funcţia - **ftell**

Returnează poziţia curentă într-un fişier.
Sintaxa este: *position = ftell(fileID)*

Position va avea valoarea -1 dacă funcţia nu este executată cu succes.

Funcţia - **fread**

Citeşte date dintr-un fişier binar.
Sintaxele funcţiei sunt:

A = fread(fileID)

Citeşte datele dintr-un fişier binar, le salvează într-un vector coloană **A**, şi poziţionează pointerul din fişier în dreptul marker-ului de sfârşit de fişier (eof).

A = fread(fileID, sizeA)

Va citi un număr de **sizeA** elemente din fişierul cu indicatorul **fileID**, şi va poziţiona pointerul din fişier după ultimul element citit. **sizeA** poate fi un număr întreg sau un vector **[m,n].**

A = fread(fileID, sizeA, precision)

În acest caz se poate defini modul de interpretare al valorilor citite în funcţie de parametrii specificaţi pentru **precision.** Valoarea nativă este 'uint8=>double'.

Funcţia - **fread**

A = fread(fileID, sizeA, precision, skip)

Dacă se adaugă şi parametrul opţional **skip**, funcţia va sări peste **skip** bytes după citirea fiecărei valori.

A = fread(fileID, sizeA, precision, skip, machineformat)

Un alt parametru opţional este **machineformat** care se referă la modul în care trebuie citite datele, acesta fiind prezentat şi mai sus.
[A, count] = fread(...)

Dacă în stânga egalului se adaugă şi parametrul opţional **count** acesta va conţine numărul de elemente citite şi salvate în **A**.

Funcţia - **fwrite**

Scrie date într-un fişier binar. Sintaxele funcţiei sunt:
fwrite(fileID, A)

Scrie datele din vectorul **A** în fişierul cu identificatorul **fileID**, coloană cu coloană.

fwrite(fileID, A, precision)
fwrite(fileID, A, precision, skip)
fwrite(fileID, A, precision, skip, machineformat)
count = fwrite(...)

Parametrul opţional **skip** va arăta numărul de bytes peste care programul va sări înainte de a scrie următoarea valoare.
Count va conţine numărul de elemente din A care au fost scrie în fişier.
Ceilalţi parametri sunt identici cu cei ai funcţiilor prezentate anterior.

Funcţia - **fscanf**

Funcţie care citeşte date dintr-un fişier text.
Funcţia are trei sintaxe principale:
A = fscanf(fileID, format)

Funcţia citeşte şi converteşte datele dintr-un fişier text într-un vector de valori A, pe coloană. Pentru conversia datelor, funcţia **fscanf** foloseşte un set de specificatori de format, care vor fi prezentaţi mai jos.

Fauncţia **fscanf** va citi pe rând datele, respectând formatul definit, până la sfârşitul fişierului, poziţionând pointerul pe marcajul de sfârşit de fişier. Dacă funcţia nu poate interpreta datele cu specificatorii de format definiţi, se vor citi datele din fişier până în momentul în care informaţia nu mai poate fi convertită moment în care funcţia se va opri.

A = fscanf(fileID, format, sizeA)

Opţiunea **sizeA** va determina citirea a unui număr egal cu **sizeA** de elemente, pointerul din fişier oprindu-se după ultimul element citit. Ca şi anterior, **sizeA** poate fi un **întreg** sau un vector de forma **[m,n]**.

Funcţia - **fscanf**

[A, count] = fscanf(...)

Va returna în **count** numărul de elemente care au fost citite cu succes de funcţie.

Specificatorii de format
Aceşti specificatori sunt foarte asemănători ca şi utilizare cu cei folosiţi în limbajul C şi permit o abordare (atât la citire cât şi la scriere) mult mai bine organizată asupra datelor. Întrucât aceştia se folosesc atât pentru citirea cât şi pentru scrierea datelor se vor prezenta la finalul acestui tabel.

Funcţia - **fprintf**

Funcţie folosită pentru scrierea datelor în fişiere sau în fereastra de comandă într-o structură bine definită. Funcţia are trei sintaxe:
fprintf(fileID,formatSpec,A1,...,An)

Funcţia va scrie, în fişierul cu indicatorul **fileID**, folosind formatarea definită în **formatSpec**, toate elementele vectorilor A1, …, An, mergând pe coloană.

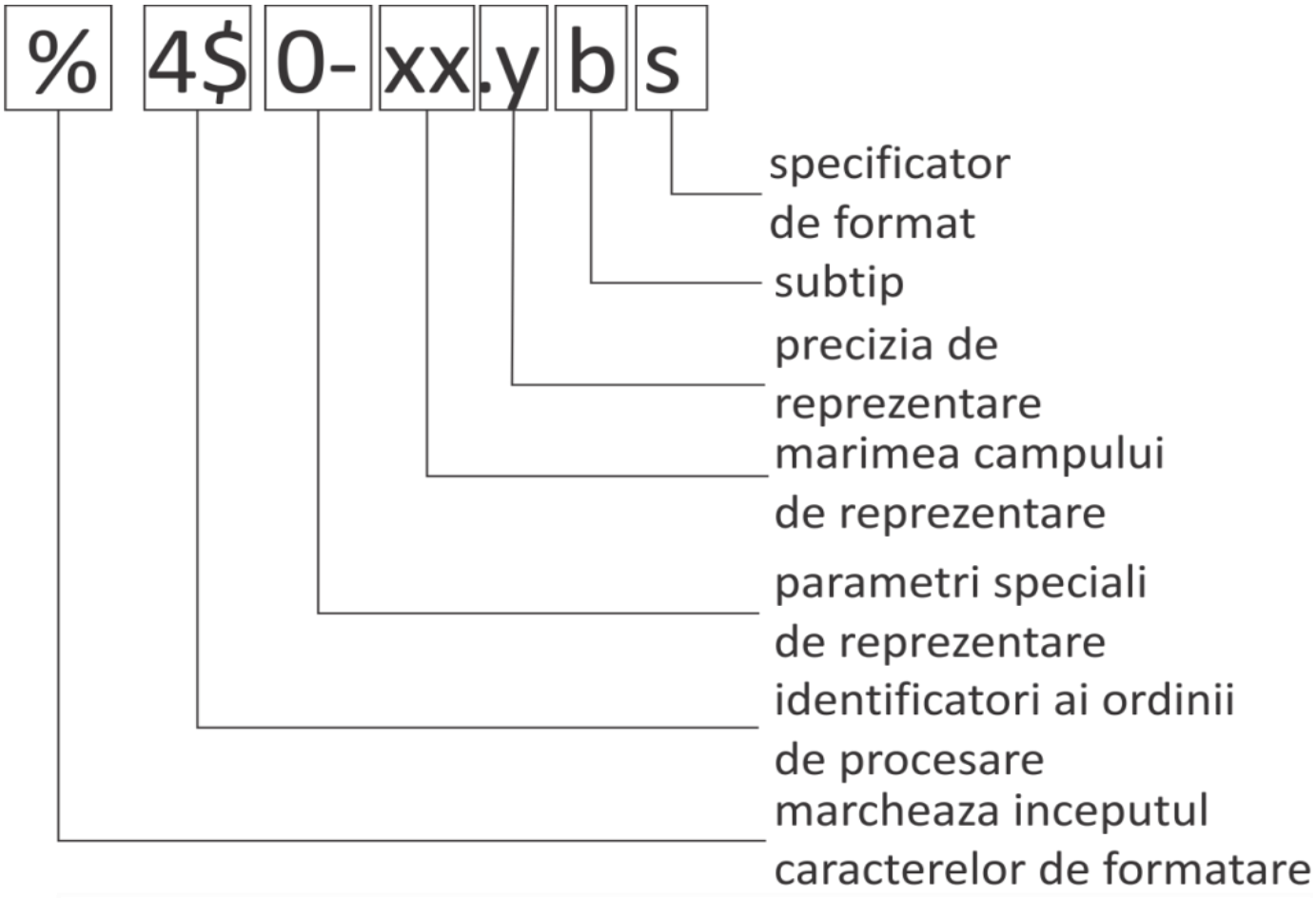
nbytes = fprintf(fileID,formatSpec,A1,...,An)
Sintaxa a doua va salva, în plus faţă de prima numărul de bytes pe care îi scrie funcţia **fprintf** în fişier.

fprintf(formatSpec,A1,...,An)
Această sintaxă are acelaşi efect ca şi prima, însă informaţia este scrisă în fereastra de comandă în loc să fie scrisă într-un fişier.

Specificatori de format pentru fscanf		
Tipul de dată	Notaţie	Detalii
Întreg cu semn (Signed integer)	%d	Număr în baza 10.
	%i	Baza definită de valoarea citită. Baza predefinită este 10, dar dacă numărul începe cu **0x** sau **0X** va fi tratat ca un număr în baza 16, şi dacă începe cu **0**, va fi tratat ca număr în baza 8.
	%ld sau %li	Valori pe 64 de biţi, în bazele 10, 8 sau 16.
Întreg fără semn (Unsigned integer)	%u	Număr în baza 10.
	%o	Număr în baza 8.
	%x	Număr în baza 16.
	%lu, %lo, %lx	Valori pe 64 de biţi în cele trei baze.
Numere reale	%f	Orice tip de număr real, inclusiv valorile nenumerice: Inf, -Inf, NaN, -NaN
	%e	
	%g	
Şiruri de caractere	%s	Se citeşte un şir de caractere, până la întâlnirea caracterului spaţiu.
	%c	Citeşte un singur caracter, inclusiv spaţiu. Pentru a citi mai multe caractere trebuie specificată lungimea câmpului.
	%[…]	Citeşte doar caracterele specificate între paranteze până la întâlnirea unui alt caracter sau a unui spaţiu.

Specificatori de format pentru fprintf		
Tipul de dată	Notaţie	Detalii
Întreg cu semn (Signed integer)	%d sau %i	Număr în baza 10.
Întreg fără semn (Unsigned integer)	%u	Număr în baza 10.
	%o	Număr în baza 8.
	%x	Număr în baza 16, folosind litere mici (a-f) pentru cifrele mai mari ca 10.
	%X	Număr în baza 16, folosind litere mari (A-F) pentru cifrele mai mari ca 10.
Numere reale (Floating - point)	%f	Notaţie cu parte zecimală fixă.
	%e	Notaţie exponenţială, cu litera ’e’ mic.
	%E	Notaţie exponenţială, cu litera ’E’ mic.
	%g	Forma mai compactă care se obţine între %e şi %f fără zerouri în coadă.
	%G	Forma mai compactă care se obţine între %E şi %f fără zerouri în coadă.
	%bx sau %bX %bo %bu	Valori în baza 16, 8 sau 10, în dublă precizie.
	%tx sau %tX %to %tu	Valori în baza 16, 8 sau 10, în simplă precizie.
Şiruri de caractere	%c	Scrie un singur caracter.
	%c	Scrie un şir de caractere.

Construcţia formatării datelor prin funcţia **fprintf**



Semnificaţia elementelor componente din formatarea parametrului **formatSpec**

% - marchează începutul parametrului;
4$ - marchează ordinea de citire a datelor dintr-o listă;
Parametri speciali de reprezentare sunt:

- ’-’ Aliniere la stânga;
- ’+’ Afişează caracterul ’+’ pentru valorile pozitive;
- ’ ’ Introduce un câmp gol (spaţiu) înaintea valorii;
- ’0’ Completează spaţiile până la dimensiunea câmpului cu zero-uri;
- ’#’ Modifică modul de reprezentare pentru conversiile numerice:
  - Pentru %0, %0x şi %0X se va tipări şi prefixul 0, 0x sau 0X;
  - Pentru %f, %e şi %E se va tipări şi punctul chiar dacă valoarea afişată este întreagă;
  - Pentru %g şi %G nu se vor şterge zero-urile şi nici punctul zecimal.

Mărimea câmpului de reprezentare este un alt parametru opţional care defineşte numărul minim de câmpuri pentru reprezentarea unei valori. Acesta poate să fie un număr întreg sau caracterul * care face trimitere spre un argument din listă.

Precizia este un număr care se introduce după semnul punct (.) şi reprezintă:

- pentru %f, %e şi %E numărul de zecimale care vor apărea în dreapta punctului;
- pentru %g şi %G numărul de cifre semnificative care să fie afişate.

Semnificaţia elementelor componente din formatarea parametrului **formatSpec**

În câmpul de formatare, **formatSpec** se poate introduce şi text, care va fi tratat ca un şir de caractere (string) care va fi reprodus exact în aceeaşi formă şi în plus se definesc o listă de caractere speciale după cum urmează:

- ’’ – introduce caracterul ghilimele;
- %% - introduce caracterul %;
- \\ - introduce caracterul \ (backslash);
- \a – generează o alarmă;
- \b – şterge ultimul caracter;
- \f – introduce un caracter gol (spaţiu);
- \n – trece pe o linie nouă;
- \r – introduce caracterul Enter (carriage return);
- \t – introduce un tab orizontal;
- \v – introduce un tab vertical;
-\xN – introduce numărul N în format hexazecimal (baza 16);
-\N – introduce numărul N în format octal (baza 8).

EXEMPLU

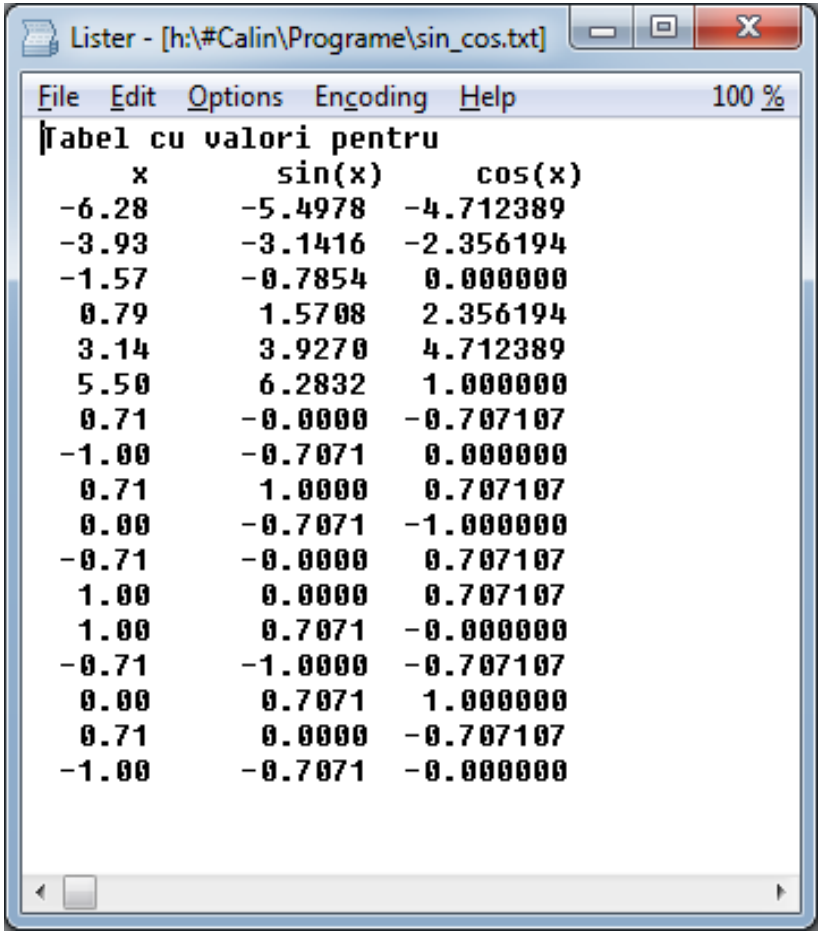
Programul generează un vector de valori pentru care determină valorile unor funcţii (sinus şi cosinus) şi le salvează într-un fişier. Apoi valorile sunt citite, salvate într-o variabilă nouă şi afişate.

```
%%program pentru scrierea valorilor intr-un fisier
clear, clc
x=-2*pi:pi/4:2*pi;
A=[x cos(x) sin(x)];
fileID=fopen('sin_cos.txt','w');
fprintf(fileID,'Tabel cu valori pentru \n     x       sin(x)
cos(x)\n');
fprintf(fileID,'%6.2f  %10.4f %10.6f\n',A);
fclose(fileID);

%%citirea valorilor

fileID=fopen('sin_cos.txt','r');
frewind(fileID);
titlu_linie1=fgets(fileID);
titlu_linie2=fgets(fileID);
AA=fscanf(fileID,'%6f %10f %10f');
AA=AA';
fclose(fileID);

%afisarea valorilor citite

disp(titlu_linie2);
fprintf('%6.2f  %10.4f %10.6f\n',AA);
```

REZULTAT

x	sin(x)	cos(x)
-6.28	-5.4978	-4.712389
-3.93	-3.1416	-2.356194
-1.57	-0.7854	0.000000
0.79	1.5708	2.356194
3.14	3.9270	4.712389
5.50	6.2832	1.000000
0.71	-0.0000	-0.707107
-1.00	-0.7071	0.000000
0.71	1.0000	0.707107
0.00	-0.7071	-1.000000
-0.71	-0.0000	0.707107
1.00	0.0000	0.707107
1.00	0.7071	-0.000000
-0.71	-1.0000	-0.707107
0.00	0.7071	1.000000
0.71	0.0000	-0.707107
-1.00	-0.7071	-0.000000

Conţinutul fişierului text creat în programul anterior (sin_cos.txt)



Observaţii

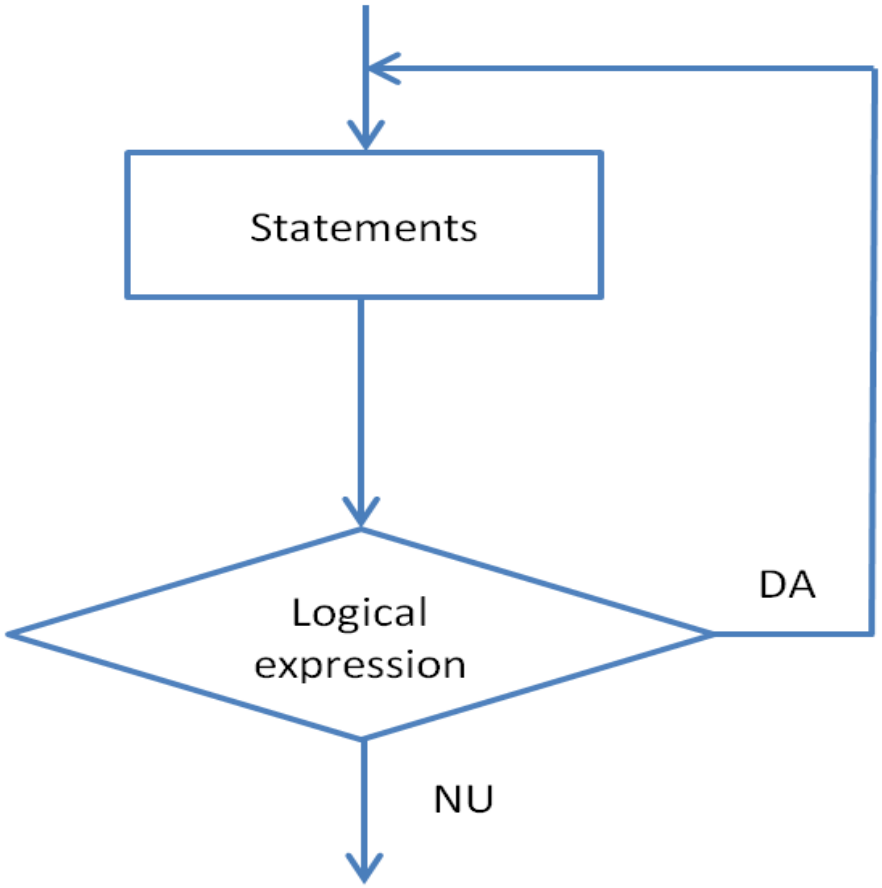
- funcţia **fscanf** nu permite definirea unei precizii pentru partea zecimală a numerelor;

- funcţia **fprintf** permite atât salvarea datelor într-un fişier cât şi reprezentarea lor în fereastra de comandă;

- datorită similitudinilor cu funcţia printf din limbajul C şi a parametrilor care pot fi configuraţi, funcţia **fprintf** se recomandă pentru afişarea/salvarea structurată a datelor de ieşire din cadrul unui program.

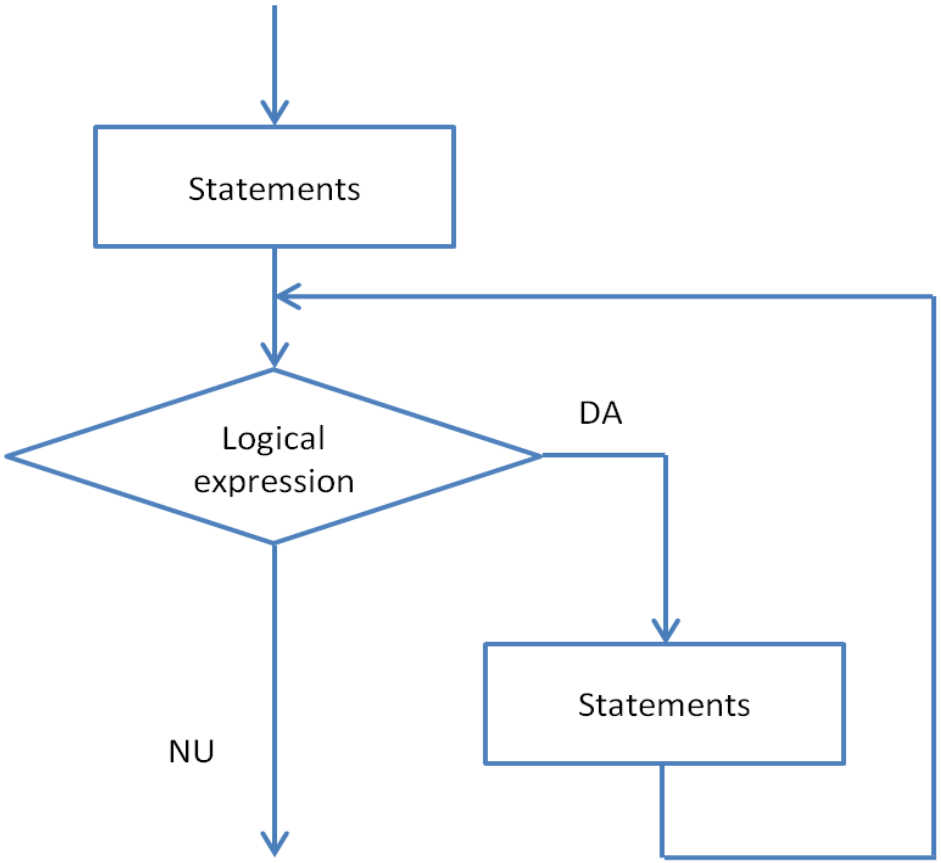
Iteraţia cu număr necunoscut de paşi cu test final

Schema logică a instrucţiunii repetitive cu test final **do-while**



În cazul instrucţiunii cu test final blocul **statements** se va executa cel puţin o dată, indiferent de valoarea de adevăr a condiţiei **logical expression**.

Schema logică echivalentă instrucţiunii repetitive cu **test final do-while**



Soluţia propusă introduce blocul de instrucţiuni **Statements** în afara ciclului **while**, cu un pas înainte de evaluarea condiţiei de ciclare (**logical expression**) ceea ce asigură execuţia, cel puţin o dată, a blocului de instrucţiuni din corpul instrucţiunii chiar dacă condiţia de ciclare este falsă.

Schema logică echivalentă instrucţiunii repetitive cu **test final do-while**

`instr1;`
`instr2;`
`while (condiţie)`
`  instr1;`
`  instr2;`
`end`
	`while (1)`
	`  instr1;`
	`  instr2;`
	`  if (~condiţie)`
	`  break;`
	`  end`
	`end`

Soluţia 1. Scrierea blocului de instrucţiuni de două ori	Soluţia 2. Mutarea condiţiei de ciclare la finalul instrucţiunii **while**

Soluţia 2, creează un ciclu infinit prin utilizarea sintaxei **while (1)** şi astfel asigură intrarea în corpul instrucţiunii **while.** Condiţia reală din **while** se mută într-o structură decizională **if** care prin testarea **condiţiei negate** (~condiţie) va genera o ieşire forţată din corpul instrucţiunii **while** când condiţia de ciclare nu mai este adevărată.

EXEMPLU

Se prezintă în continuare un algoritm care citeşte, solicită introducerea valorii unui an şi stabileşte dacă anul respectiv este bisect sau nu. De asemenea, programul permite repetarea algoritmului de câte ori doreşte utilizatorul.

```
%%Program an bisect cu repetitie
clear, clc;
while (1)

    %inceput bloc instructiuni
    an=input('Introduceti anul:');

    if (~rem(an,4))
        disp ('Anul este bisect!!')
    else
        disp('Anul nu este bisect!!')
    end
    raspuns = input('Doriti introducerea altui an? D/N [N]','s');
    if isempty(raspuns)
        raspuns='N';
    end
    %sfarsit bloc instructiuni
    if ~(raspuns == 'D')
        break;
    end
end

%%Sfarsit program
```

Rulare program

Introduceti anul:1246
Anul nu este bisect!!
Doriti introducerea altui an? D/N [N]D
Introduceti anul:6543
Anul nu este bisect!!
Doriti introducerea altui an? D/N [N]D
Introduceti anul:1980
Anul este bisect!!
Doriti introducerea altui an? D/N [N]N

OBSERVATIE
La rularea programului, se va solicita introducerea unui valorii unui an şi programul va verifica (prin divizibilitatea cu 4) dacă anul este bisect, întrebând apoi utilizatorul dacă doreşte o nouă rulare. La introducerea oricărui alt caracter în afară de (’D’) programul va considera răspunsul negativ şi va ieşi din ciclul **while**. Dacă în acest program s-ar fi optat pentru prima soluţie, toate instrucţiunile din blocul de instrucţiuni ar fi trebuit scrise de două ori, ceea ce ar fi complicat nejustificat algoritmul.

Intrebari