Lucrarea de laborator nr. 4

I. Scopul lucrării

Elemente de programare în MAPLE.

II. Conținutul lucrării

1. Numere, şiruri şi identificatori.
2. Structuri de date.
3. Atribuirea. Decizia. Structuri repetitive.
4. Proceduri în MAPLE.

III. Prezentarea lucrării

III.1. Numere, şiruri şi identificatori.

Constantele numerice din MAPLE sunt de trei tipuri:

- întregi
- raționale
- în virgulă mobilă

Constantele întregi sunt şiruri de cifre zecimale (0..9) eventual precedate de un semn (+,-) reprezentând un număr întreg. Numărul maxim de cifre permise este dependent de sistem, dar în general este mai mare de 500 000.

Exemple de constante întregi:

```
> 0;
                              0
> 123;
                            123
> -6789;
                          -6789
> 12345678912345678912345678;
          12345678912345678912345678
```

Constantele raționale utilizează operatorul de împărțire / pentru a separa numărătorul de numitor. Astfel m/n cu m şi n constante întregi reprezintă numărul rațional $\frac{m}{n}$.

Exemple de constante raționale:

```
> 2/3;
```

$$\frac{2}{3}$$

```
> -6/7;
```

$$\frac{-6}{7}$$

```
> 4/6;
```

$$\frac{2}{3}$$

```
> 4/2;
```

$$2$$

```
> -39/13;
```

$$-3$$

Se observă că MAPLE face automat simplificarea fracțiilor.

Reprezentarea unei ***constante în virgulă mobilă*** conține în general câmpurile următoare:

- partea întreagă
- punctul zecimal
- partea fracționară

e sau E şi un exponent cu semn (opțional);
Se poate omite partea întreagă sau partea fracționară, dar nu amândouă. De asemenea, se poate omite punctul zecimal sau litera e(E) şi exponentul, dar nu amândouă.

Exemple de constante în virgulă mobilă:

```
>  2.3;
```

$$2.3$$

```
> 678.96e-9
```

$$.67896\ 10^{-6}$$

```
> .1234;
```

$$.1234$$

```
> 123E56;
```

$$.123\ 10^{59}$$

```
> 1.;
                                    1.
```

Constante în virgulă mobilă pot fi obținute şi cu comanda **Float**. Această comandă are forma

Float(mantisa,exponent);

şi întoarce mantisa*10 ^exponent.

```
> Float(123,56);

                                   59
                                .123 10
```

Expresiile aritmetice cu operanzi constante întregi sau raționale sunt evaluate exact în MAPLE (rezultatul este o constantă rațională sau o constantă întreagă).

Exemple:

```
> 1/3+4/5;
                                   17
                                   --
                                   15
> 1/3+8;
                                   25
                                   --
                                    3
> 1/3+2/3;
                                    1
```

În cazul în care expresia conține constante în virgulă mobilă, atunci constantele întregi şi cele raționale (care apar eventual în expresie) sunt convertite în virgulă mobilă (sunt aproximate cu constante în virgulă mobilă). Rezultatul expresiei este în acest caz o constantă în virgulă mobilă.

Exemple:

```
> 1/3.+4/5;
                          1.133333333
> 1./3+8;
                          8.333333333
> 1/3+2/3.;
                          1.000000000
> 20+45.75e-2;
                           20.4575
```

Orice număr real nenul x poate fi scris sub formă normalizată, în baza 10:

$$x = \pm m\ 10^{p}$$

cu $0{,}1 \le m < 1$, (m = mantisa). În calcule se reține de obicei un număr finit de cifre zecimale ale mantisei. Numărul de cifre care se reține se numeşte ***număr de cifre semnificative***. Numărul de cifre semnificative poate fi controlat în

MAPLE cu ajutorul variabilei globale **Digits**. Valoarea implicită pentru pentru digits este 10.

Exemple:

```
> 1./7;
                        .1428571429
> Digits:=20;
                      Digits := 20
> 1./7;
                  .14285714285714285714
```

Deci MAPLE poate lucra în virgulă mobilă cu o precizie teoretic "infinită".

Pentru a determina evaluarea unei expresii în virgulă mobilă (chiar dacă toți operanzii din expresie sunt întregi sau raționali) se poate folosi comanda **evalf**.

evalf(expresie)

determină evaluarea expresiei la o valoare în virgulă mobilă, cu numărul de cifre semnificative stabilit de variabila Digits.

evalf(expresie,n)

determină evaluarea expresiei la o valoare în virgulă mobilă, utilizând n de cifre semnificative (valoarea variabilei Digits nu este afectată).

Exemple:

```
> evalf(1/7);
                        .1428571429
> evalf(1/7,20);
                  .14285714285714285714
> evalf(Pi);
                        3.141592654
> evalf(Pi,30);
           3.14159265358979323846264338328
```

Există o întreagă familie de funcții de evaluare numerică şi algebrică a expresiilor:

- **eval** – evaluează în întregime o expresie
- **evala**– evaluează algebric o expresie
- **evalf**– evaluează numeric o expresie
- **evalb**– evaluează boolean o expresie
- **evalm**– evaluează matriceal o expresie
- **evalc**– evaluează în mulțimea numerelor complexe o expresie

În MAPLE un ***şir de caractere*** (**string**) constă dintr-o succesiune de caractere cuprinse între apostrofuri întoarse (backquote) (`). Punctul (.) reprezintă operatorul de concatenare pentru şirurile de caractere în MAPLE.

Exemple:

```
> `Acesta este un string in MAPLE`;
        Acesta este un string in MAPLE
```

```
> `1+2=?`;
                          1+2=?
> `acesta este. un string`;
                acesta este. un string
> `acesta este`.` un string`;
                 acesta este un string
```

Un ***identificator*** în MAPLE este un şir de caractere alfabetice (A-Z, a-z), cifre (0-9) şi caracterul _ (liniuţa de subliniere, underline), şir în care primul caracter este un caracter alfabetic (A-Z, a-z). Un identificator nu poate conţine mai mult de 499 de caractere. MAPLE este case-sensitive, ceea ce însemnă că identificatorul nume este diferit de identificatorul Nume. Identificatorii nu trebuie incluşi între (`). MAPLE conţine un număr de ***identificatori predefiniţi*** (identificatori rezervaţi). O listă a acestora poate fi obţinută cu comanda

```
>? ininame
```

sau

```
> help (`ininame`);
```

III.2. Structuri de date

Listele (**lists**) în MAPLE sunt şiruri ordonate de expresii, separate între ele prin virgulă şi incluse între paranteze drepte []. Ordinea expresiilor este dată de poziţia în care apar în listă. Dacă L este o listă **L[i]** desemnează elementul de pe poziţia i. ***Lista vidă*** este desemnată prin **[]**. Se pot efectua următoarele operaţii cu liste:

- ***extragerea*** din lista L a elementelor de poziţia i până la poziţia j: L(i..j) sau op(i..j,L).;
- ***adăugarea*** unui element x la lista L: [x,op(L)] (adaugă elementul pe prima poziţie), [op(L),x] (adaugă elementul pe ultima poziţie);
- ***modificarea*** elementului de pe poziţia i: subsop(i=x,L) sau L[i]:=x;
- **eliminarea** elementului de pe poziţia i: subsop(i=NULL,L);

Exemple:

```
> L:=[1,2,3,4];
                  L := [1, 2, 3, 4]
> L[2];
                          2

> L[2]:=5;
```

```
                          L[2] := 5
> L;
                        [1, 5, 3, 4]
> L[2..4];
                         [5, 3, 4]
> op(2..4,L);
                          5, 3, 4
> L1:=[6,op(L)];
                  L1 := [6, 1, 5, 3, 4]
> L2:=[op(L),6];
                  L2 := [1, 5, 3, 4, 6]
> subsop(4=7,L2);
                     [1, 5, 3, 7, 6]
> L2;
                     [1, 5, 3, 4, 6]
> subsop(4=NULL,L2);
                      [1, 5, 3, 6]
> L2;
                     [1, 5, 3, 4, 6]
```

Mulțimile (**sets**) în MAPLE sunt şiruri neordonate de expresii, separate între ele prin virgulă şi incluse între acolade {}. Duplicatele sunt eliminate. ***Mulțimea vidă*** este desemnată prin **{}**. Se pot efectua următoarele operații cu mulțimi:

- reuniune: operatorul union
- intersecție: operatorul intersect
- diferență: operatorul minus

Exemple:

```
> M:={red, green, blue};
               M := {red, green, blue}
> S:={1,2,1,3,2};
                     S := {1, 2, 3}
> M union S;
             {1, 2, 3, red, green, blue}
> S minus {2};
                          {1, 3}
> S intersect {2,3,7};
                          {2, 3}
```

Tablourile (**tables**) în MAPLE sunt structuri de date ai căror membri sunt indexați.

Exemple:

```
> t:=table([(culoare1)=red,(culoare2)=green,
  (culoare3)=blue]);
    t := table([
```

```
          culoare1 = red
          culoare2 = green
          culoare3 = blue
          ])
> t[culoare2];
                                green
```

Un tablou cu zero sau mai multe dimensiuni, pentru care fiecare dimensiune are domeniu întreg se numeşte în MAPLE **array**. Pentru a crea un array se poate apela funcţia array sub forma:

array(domeniile de indexare, listă de iniţializare)

Parametrii sunt opţionali şi pot apărea în orice ordine.

Exemple:

```
> v := array(1..4);
                v := array(1 .. 4, [])
> v[2];
                          v[2]
> v[2]:=3;
                       v[2] := 3

> evalm(v);
                 [v[1], 3, v[3], v[4]]
> A := array(1..2,1..2);
            A := array(1 .. 2, 1 .. 2, [])
> A[1,2] := x;
                     A[1, 2] := x
> A[1,1];
                       A[1, 1]
> A[1,2];
                          x
> evalm(A);
                  [ A[1, 1]   x    ]
                  [                ]
                  [ A[2, 1] A[2, 2] ]

> A := array(1..2,1..2, [ [1,x], [x,x^2] ] );
```

$$A := \begin{bmatrix} 1 & x \\ x & x^2 \end{bmatrix}$$

Matricele (**matrix**) în MAPLE sunt tablouri bidimensionale cu indexate de la 1. Cu alte cuvinte un apel matrix(m,n, listă de inițializare) este echivalent cu array(1..m,1..n, listă de inițializare).

Exemple:

```
> M:=matrix(3,2,[[1,2],[3,4],[5,6]]);
```

$$M := \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
> M[1,2];
```

$$2$$

Pachetul **linalg** conține comenzi pentru multe operații cu matrice.

III.3. Atribuirea. Decizia. Structuri repetitive.

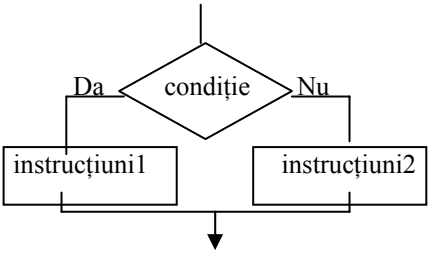
Atribuirea are forma

x:=v:

Efectul acestei instrucțiuni constă în evaluarea expresiei v pentru valorile curente ale variabilelor pe care le conține şi înscrierea rezultatului în locația de memorie rezervată variabilei x

Decizia are forma:

if condiție **then** instrucțiuni1 **else** instrucțiuni2 **fi**;

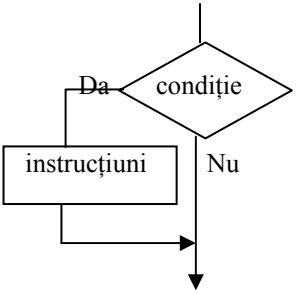


Condiția este o expresie logică (formată cu operatori logici sau relaționali). Modul de execuție al deciziei (precum rezultă din subschema logică de mai sus) este următorul:

1. se evaluează condiția
2. dacă rezultatul este adevărat se execută instrucțiuni1, în caz contrar se execută instrucțiuni2.
3. se trece la comanda care urmează după decizie

În cazul în care **else** lipseşte se foloseşte forma simplificată:

if condiție **then** instrucțiuni **fi**;



1. se evaluează condiția
2. dacă rezultatul este adevărat se execută instrucțiuni
3. se trece la comanda care urmează după decizie

Un extra element **elif** (ținând loc de **else**+**if**) poate fi adăugat în decizie, obținând:

if/then/elif/then…/else/fi

Exemple:

```
> a := 3; b := 7;
                          a := 3
                          b := 7
> if (a > b) then a else b fi;
                            7
> if (a > b) then c:=7 fi;
> c;
                            4
> if (a > b) then c:=7 elif (a<b) then c:=9 fi;
                          c := 9
```

Există două instrucțiuni repetitive în MAPLE: **for** şi **while**. For are mai multe forme:

(1) for i **from** ei **by** p **to** ef **do** instrucțiuni **od**;

unde i este variabila de contorizare, p este pasul cu care se face incrementarea (decrementarea), iar ei (respectiv ef) este o expresie care determină valoarea inițială (respectiv finală) a contorului. Modul de execuție al acestei instrucțiuni este următorul:

1. se execută atribuirea i : = ei

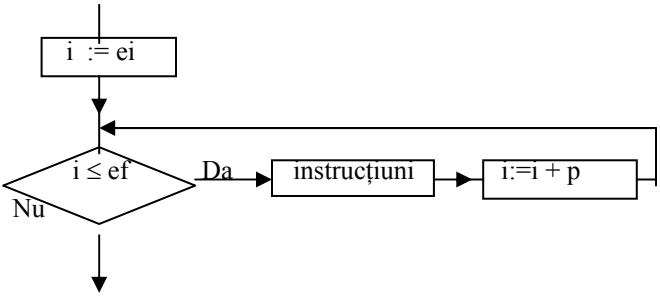
2. se evaluează condiția i ≤ ef dacă p > 0 (sau i ≥ ef dacă p < 0), şi dacă este îndeplinită această condiție se trece la pasul 3, altfel se trece la pasul 5

3. se execută instrucțiuni

4. se execută atribuirea i := i + p

5. se execută comanda care urmează după for

Comanda este echivalentă cu următoarea subschemă logică (pentru p >0):



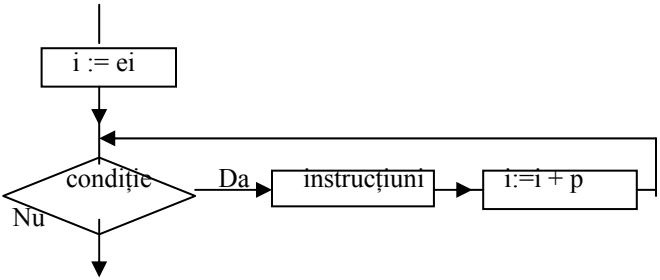
Construcțiile from ei şi by p poate lipsi, caz în care ei se ia 1 şi pasul se consideră egal cu 1.

(2) **for** i **from** ei **by** p **while** condiție
do instrucțiuni **od**;

Modul de execuție al acestei instrucțiuni este următorul:

1. se execută atribuirea i : = ei
2. se evaluează condiția trecută după while, şi dacă este îndeplinită, se trece la pasul 3, altfel se trece la pasul 5
3. se execută instrucțiuni
4. se execută atribuirea i := i + p
5. se execută comanda care urmează după for

Comanda este echivalentă cu următoarea subschemă logică:



Ca şi înainte construcțiile from ei şi by p poate lipsi, caz în care ei se ia 1, iar pasul se consideră egal cu 1. Condiția este dată printr-o expresie booleană.

Ambele clause **to** şi **while** pot fi prezente în instrucțiunea for:

(3) **for** i **from** ei **by** p **to** ef **while** condiție
do instrucțiuni **od**;

În acest caz

1. se execută atribuirea i : = ei
2. se evaluează condiția i ≤ ef dacă p > 0 (sau i ≥ ef dacă p < 0), și condiția trecută după while; dacă amândouă sunt îndeplinite se trece la pasul 3, altfel se trece la pasul 5
3. se execută instrucțiuni
4. se execută atribuirea i := i + p
5. se execută comanda care urmează după for

În cazul următoarei instrucțiuni for contorul i parcurge toate elementele unei liste sau unei mulțimi (expr):

(4) **for** i **in** expr **while** condiție
do instrucțiuni **od**;

Exemple:

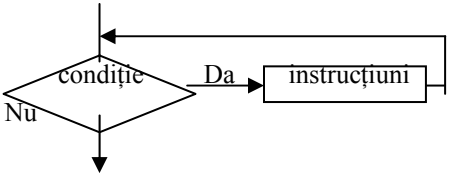
```
> for i from 6 by 2 to 10 do  print(i)  od;
                            6
                            8
                           10
> suma := 0;
> for i from 11 by 2 while i < 15 do
>    suma := suma + i
> od;
                         suma := 0
                        suma := 11
                        suma := 24
> L:=[1,5,3];
                     L := [1, 5, 3]
> suma:=0;
> for z in L do
> suma:=suma+z
> od;
                         suma := 0
                         suma := 1
```

```
suma := 6
suma := 9
```

Ciclu cu test inițial are forma:

while condiție **do** instrucțiuni **od**;

Testul pentru repetarea calculelor se face înaintea execuției grupului de comenzi care trebuie repetate. Dacă este îndeplinită condiția, se execută instrucțiunile după care se reevaluează condiția. În caz contrar, se trece la comanda care urmează după ciclul cu test inițial. Subschema logică echivalentă este următoarea:



Condiție reprezintă o expresie booleană.

Exemple:

```
> x:=234;
                        x := 234
> while x>0 do x:=iquo(x,10,'r');print(r); od;
                        x := 23
                           4
                         x := 2
                           3
                         x := 0
```

III.4. Proceduri în MAPLE.

În principal, necesitatea subprogramelor se datorează faptului că de multe ori algoritmul prevede executarea aceloraşi instrucțiuni pentru date diferite. Grupul de instrucțiuni care se repetă poate constitui o unitate distinctă căreia i se dă un nume şi un set de parametri. Ori de câte ori va fi necesară execuția acestui grup de instrucțiuni se specifică numele şi parametrii care actualizează grupul de instrucțiuni (astfel se scurtează dimensiunea şi creşte claritate programului). Grupul de instrucțiuni se numeşte ***procedură*** (**procedure**) în MAPLE.

Forma unei proceduri este:

```
nume:=proc (param1, param2,…)
```

```
local lista declaraţii locale;
global lista declaraţii globale;
options listă opţiuni;
description descriere;
instrucţiuni
end
```

Nu toate elementele de mai sus sunt obligatorii. Dacă este necesar ca procedura să întoarcă o valoare, se poate folosi apelul

RETURN(v)

în şirul de instrucţiuni din corpul procedurii.

Parametrii care apar în scrierea unei proceduri se numesc ***parametrii formali***, ei având un rol descriptiv (un parametru formal este o variabilă al cărei nume este cunoscut, dar al cărei conţinut nu este precizat decât în momentul execuţiei). În cadrul listei parametrii formali sunt separaţi prin virgulă. Numele procedurii (nume) este un identificator MAPLE Apelul unei proceduri se face cu comanda:

nume (listă parametrii actuali)

parametrii actuali fiind expresii despărţite între ele prin virgulă în cadrul listei. În momentul execuţiei parametrii actuali substituie parametrii formali. Un apel de procedură determină următoarele acţiuni:

- se stabileşte corespondenţa între argumente şi parametrii
- se execută instrucţiunile subprogramului, până când se ajunge la **end** sau la o instrucţiune **RETURN**. Efectul acestor instrucţiuni (**end** şi **RETURN**) este întoarcerea în unitatea de program în care a avut loc apelul, şi anume la instrucţiunea ce urmează imediat acestui apel (precizăm că o procedură poate apela la rândul său o altă procedură). Un apel de procedură este corect dacă între parametrii actuali şi cei formali există o corespondenţă atât ca număr, cât şi ca tip şi organizare.

Exemplu:

Să presupunem că se dă un număr întreg pozitiv x, şi se cere lista cifrelor corespunzând reprezentării binare a lui x. Procedura următoare rezolvă această problemă.

```
> lbinar:=proc(x)
    local y,L;
    y:=x; L:=[];
      while y>0 do
         L:=[irem(y,2,'c'),op(L)];y:=c;
      od;
```

```
          RETURN(L);
end;
> lbinar(27);
                      [1, 1, 0, 1, 1,1]
> lbinar(32);
                     [1, 0, 0, 0, 0, 0]
```

Procedura lbinarfr de mai jos întoarce lista primelor n cifre ale reprezentării binare a unui număr x, cu proprietatea $0 \le x < 1$.

```
> lbinarfr:=proc(x,n)
      local y,i,L;
      y:=x;L:=[];
      for i from 1 to n do y:=y*2;
           L:=[op(L),floor(y)];y:=frac(y)od;
      RETURN(L);
end;
> lbinarfr(0.15,10);
           [0, 0, 1, 0, 0, 1, 1, 0, 0, 1]
```

Procedura lbin întoarce lista cifrelor binare corespunzătoare unui număr real x. Pentru partea fracționară a numărului se rețin n cifre. Procedura lbin apelează procedurile precedente.

```
> lbin:=proc(x,n)
      local y,L1,L2,L;
      y:=abs(x);
      L1:=lbinar(floor(y));
      L2:=lbinarfr(frac(y),n);
      L:=[L1,L2];
      RETURN(L);
      end;
      > lbin(23.15,10);
          [1, 0, 1, 1, 1],
                     [0, 0, 1, 0, 0, 1, 1, 0, 0, 1]]
```