3. **Subprograme predefinite**

3.1. Subprograme. Mecanisme de transfer prin intermediul parametrilor

3.2. Proceduri și funcții predefinite

5. **Fisiere text**

5.1. Fișiere text. Tipuri de acces

5.2. Proceduri si funcții predefinite pentru fișiere text

Prejmerean Vasile 9:00~10:30

7. Subprograme definite de utilizator

7.1. Proceduri și funcții

- declarare și apel
- parametri formali și parametri efectivi
- parametri transmiși prin valoare, parametri transmiși prin referință
- variabile globale și variabile locale, domeniu de vizibilitate

7.2. Proiectarea modulară a rezolvării unei probleme

Lazar Ioan 10:30~12:00

3. Subprograme predefinite

3.1. Subprograme.

Mecanisme de transfer prin intermediul parametrilor

- **Subprograme** Pascal :
  - proceduri ***Procedure***,
  - functii ***Function***.

- Parametri (tipuri/clasificari),
- Vizibilitate,
- Parametri de tip Functie, Procedura,
- Apelul recursiv (subpr. recursive),
- Definire simultana (*Forward*).

Facultativ

Subalgoritmi: rezolvă o anumită subproblemă

Apel:

[***Cheamă***] *Nume_Subalgoritm (Lista_parametri_actuali);*

Def.:

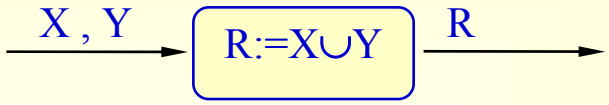
Subalgoritmul *Nume_Subalgoritm (Lista_parametri_formali)* ***Este*** *: {Antet}*

. . . { Corp subalgoritm }

Sf_Subalgoritm. *{ sau* ***Sf***_*Nume_Subalgoritm. }*

Parametri de Intrare → Expresii
Parametri de iesire → Variabile

Exemplu: *pentru trei mulţimi date A, B şi C calculăm $A \cup B$, $A \cup C$ şi $B \cup C$*



Algoritmul Reuniuni Este :
Date A,B,C;
Cheamă Reuniune (A,B,R1);
Cheamă Reuniune (A,C,R2);
Cheamă Reuniune (B,C,R3);
Rezultate R1;
Rezultate R2;
Rezultate R3;
Sf_Algoritm.

Subalgoritmul Reuniune (X,Y, R) Este :
R:=X;
Pentru fiecare $y \in Y$ Execută
Dacă $y \notin X$ Atunci $R:=R \cup \{y\}$
Sf_Dacă
Sf_Pentru
Sf_Reuniune.

Subalgoritmul Reuninune determină mulţimea $R = X \cup Y$ astfel :
$R := X \cup (Y \setminus X)$, adică depune în reuniune mai întâi elementele din mulţimea X, la care apoi mai adaugă acele elemente din Y care nu aparţin lui X.

Functii: ~ Subalgoritm + Val. functiei

Apelul unei funcţii:
se face scriind într-o expresie numele funcţiei urmat de lista parametrilor actuali.

... Nume_Funcţie (Listă_parametri_actuali) ... { →expr.→instr.}

Def.:
Funcţia *Nume_Funcţie (Listă_parametri_formali)* ***Este*** *: { Antetul funcţiei }*
. . .
Nume_Funcţie := Expresie; *{ Corpul funcţiei }*
. . .
Sf_Funcţie. *{ sau* ***Sf*** *_Nume_funcţie.}*

Parametri de Intrare → Expresii
Parametri de iesire → Variabile

Exemple: *Există & Apart*

```
Funcţia  Există  ( b, A, n, p )  Este :
        p:=1;
        Cât_Timp (p<=n) şi (b<>ap) Execută  p:=p+1  Sf_Cât_Timp;
        Există  :=  (p<=n)
Sf_Există.
```

```
Funcţia  Apart (b,A) Este :
        p:=1;                {Card(A)=|A|}
        Cât_Timp (p<=Card(A)) şi (b<>A[p]) Execută  p:=p+1
        Sf_Cât_Timp;
        Apart  :=   (p<=Card(A))
Sf_Apart.
```

```
Funcţia  Card (A) Este :
          Card := a0
Sf_Card.
```

Dacă $y \notin X$ Atunci $R:=R \cup \{y\}$ Sf_Dacă;

→ *Dacă Not* ***Apart (y,X)*** *Atunci $R:=R \cup \{y\}$ Sf_Dacă;*

Exemplu: *determină maximul dintr-un şir X cu n componente .*

```
Funcţia  Max (X,k)  Este :
    Dacă k=1 Atunci      Max:=x1                          {Consistenţa}
         Altfel   Dacă  Max(X,k-1) < xk  Atunci Max:= xk
                                         Altfel   Max:= Max(X,k-1)
                  Sf_Dacă
    Sf_Dacă
Sf_Max.
```

```
Apelul:
    Max(X,n)
```

Exemplu: *decide dacă b aparţine primelor k elemente din şirul A .*

```
Funcţia Apart (b,A,k) Este :
    Apart := (k>0)   şi  ( Apart(b,A,k-1)  Sau  (b=ak)  )
Sf_Apart.
```

```
Apelul:
    Apart(b,A,Card(A))
```


pascal

Subprograme Pascal : Procedure, Function :

<Def_subprogram> ::= <Def_funcţie> | <Def_procedură>

<Def_funcţie> ::= <Antet_funcţie> ; <Bloc>

<Def_procedură> ::= <Antet_procedură> ; <Bloc>

<Antet_funcţie> ::= Function <Nume> [(L_p_f)] : <Tip_f>

<Antet_procedură> ::= Procedure <Nume> [(L_p_f)]

Real, Integer, Byte, Boolean, Char, String, …

***Apel**:*

P: *<Nume> [(Lista_parametri_actuali)];*

F: *… <Nume> [(Listă_parametri_actuali)] …* { → expr. → instr. }

Parametri :

<L_p_f.> ::= <spf> { ; <spf> }

<spf> ::= <sp_val> | <sp_var> | <p_functie> | <p_procedura>

<sp_val> ::= <lista_id> : <id_tip>

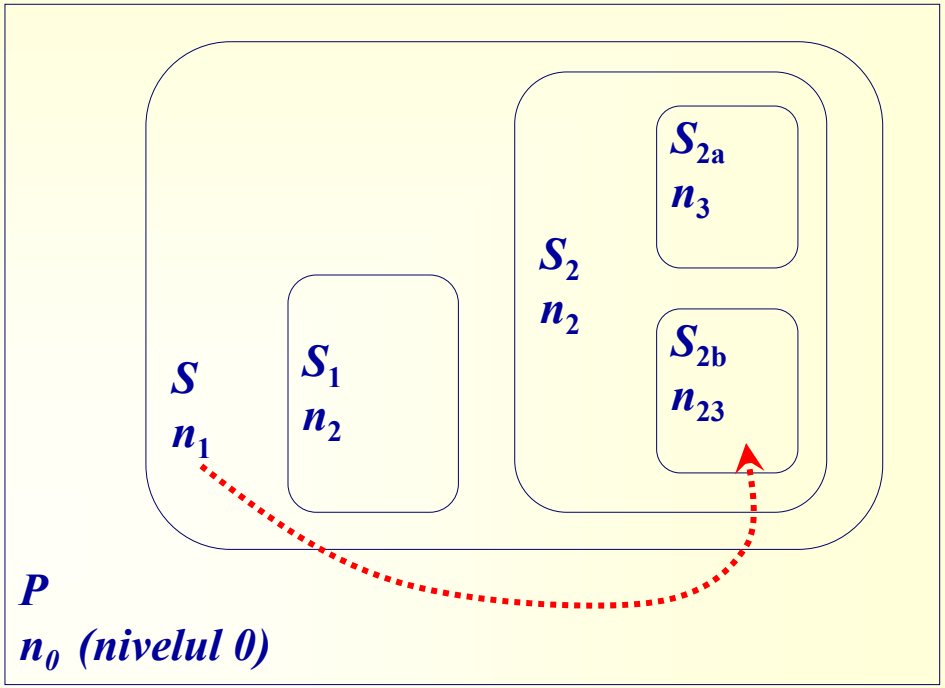
<sp_var> ::= Var <lista_id> : <id_tip>

Procedure Reuniune (A,B:Multime; Var C:Multime);

Vizibilitate :

Domeniu de vizibilitate:

- **Var.** ***locale***
- **Var.** ***globale*** **(!)**
- Proceduri,
- Functii,
- …




```
Function  Apart (b:Integer; A:Multime) : Boolean;     {Apartine b multimii A?}
Var  i,n : Integer;
Begin
            i:=1;  n:=A[0];
    While (i<=n) And (b<>A[i]) Do i:=i+1;
    Apart :=  i<=n              {Apartine, daca i<=n }
End;
```

```
Program  AuB_AuC_BuC;
Type    Multime = Array[0..100] Of Integer;
Procedure Citeste  (Var A:Multime);            { Citeste o multime de la tastatura }
Var  c,p,n : Integer;        s : String;
Begin
    Write (' Dati elementele multimii : '); Readln (s);  n:=0;
    While s<>'' Do Begin
        n:=n+1;  Val(s,A[n],p);
        If  p>0  Then Val(Copy(s,1,p-1),A[n],c) Else p:=Length(s);   Delete (s,1,p)
    End;
    A[0]:=n;        { n=Card(A)→ A[0] }
End;
Procedure Tipareste (A:Multime);               { Tipareste pe ecran o multime A }
Var  n,i : Integer;
Begin
    n:=A[0]; Write (' {');
    For i:=1 To n Do Write (A[i],',');
    Writeln (Chr(8),'}')                       { Rescrie peste ultima virgula, acolada }
End;
```

```
Function  Apart (b:Integer; A:Multime) : Boolean;      {Apartine b multimii A?}
Var  i,n : Integer;
Begin       i:=1; n:=A[0];
    While (i<=n) And (b<>A[i]) Do i:=i+1;
    Apart :=  i<=n                                    {Apartine, daca i<=n }
End;
Procedure Reuniune (A,B:Multime; Var C:Multime);          { C := A ∪ B }
Var  i,p,q,r : Integer;
Begin                C:=A;   p:=A[0]; q:=B[0];          r:=C[0];
    For i:=1 To q Do
       If  Not  Apart(B[i],A)   Then   Begin
                r:=r+1;  C[r]:=B[i]   End       C[0]:=r
End;
Var     A, B, C,   AuB, AuC, BuC : Multime;
Begin                                                  { Modulul principal }
    Citeste (A); Citeste (B); Citeste (C);
    Reuniune(A,B,AuB);  Tipareste (AuB);
    Reuniune(A,C,AuC);  Tipareste (AuC);
    Reuniune(B,C,BuC);  Tipareste (BuC);                          Readln
End.
```

Parametri de tip Function, Procedure :

Ex.: *Grafic (f,a,b); Grafic (g,c,d); …*
Integrala (f,a,b,SumeR); Integrala (g,c,s,Trapeze); …
Radac (f,a,b,eps,Coardei); Radac (g,c,d,0.01,Tangentei); …

```
Program pfp;          {$F+}.
Type   Functie    = Function   ( x : Real ) : Real ;
   Procedura  = Procedure  (   . . .   ) ;
Function  <nume_subpr> ( ... f:Functie; ...  p:Procedura; ... ) : ...
Procedure        …
Begin                    … f …          … p …                End;
```

Apelul:

```
Function  f1 (…):…                Procedure metoda1(…);…
Function  f2 (…):…                Procedure metoda2(…);…

. . .  <nume_subpr>  ( ... f1, ... ,metoda1, ... ) ...
. . .  <nume_subpr>  ( ... f2, ... ,metoda2, ... ) ...
                . . .
```

```
Program  Parametri_Functie_Procedura;        {$F+}
Type     Sir       = Array  [1..30] Of Integer;
         Functie   = Function (x:Integer)   : Integer;
         FctBool   = Function (a,b:Integer) : Boolean;
         Procedura = Procedure(Var Z:Sir; n:Integer);
Function f (x:Integer) : Integer;  Begin  f:=Sqr(x+1)-x  End;
Function g (x:Integer) : Integer;  Begin  g:=Sqr(x-1)+x  End;
Function       Cresc(a,b:Integer):Boolean; Begin       Cresc:=a<b End;
Function DesCresc(a,b:Integer):Boolean; Begin  DesCresc:=a>b End;
Procedure Ordonez  (Var X:Sir; n:Integer;  Relatie : FctBool );
Var      Ordonat : Boolean; k, i, v : Integer;
Begin                    k:=1;
  Repeat  Ordonat:=True;
          For i:=1 To n-k Do
             If Not Relatie(X[i],X[i+1]) Then    Begin
                                 v:=X[i]; X[i]:=X[i+1]; X[i+1]:=v;
               Ordonat:=False                          End;
          k:=k+1
  Until Ordonat;
End;
```

```
Procedure OrdCresc    (Var A:Sir; n:Integer);
Begin     Ordonez     (    A, n, Cresc     )  End;

Procedure OrdDesCresc (Var A:Sir; n:Integer);
Begin     Ordonez     (    A, n, DesCresc  )  End;

Procedure Gen_Ord_Tip (Var A:Sir; n:Integer;    r : Functie;
                                                Ordonare : Procedura );
Var  i:Integer;
Begin
    For i:=1 To n Do A[i]:=r(i);                          {Generare}
    Ordonare (A,n);                                       {Ordonare}
    For i:=1 To n-1 Do Write (A[i],','); Writeln(A[n])  { Tiparire }
End;

Var      X, Y : Sir;

Begin
   Gen_Ord_Tip (X,10,  f,  OrdCresc );                    { Prel X }
   Gen_Ord_Tip (Y,11,  g,  OrdDesCresc );    Readln       { Prel Y }
End.
```

Apelul recursiv :

Este permis apelul recursiv (un subprogram se autoapeleaza). Se ofera solutii simple plecand de la (avand) definitii simple.

Ex.: *Problema turnurior din Hanoi , n!=n*(n-1)! sau* 1 (Consistenta!)

Recursivitate→ Iteratie !

Ex.: *"Fiind date două numere naturale de maxim 30 de cifre, sub forma a două şiruri de caractere, să se determine şirul cifrelor sumei"* :

Formulă recursivă pentru a aduna numărul *A* de *n* cifre cu numărul *B* de *m* cifre :

$$
Ad(A,m,B,n,Tr) = \begin{cases}
Ad(B,n,A,m,Tr) & \text{Dacă } m > n & (1)\\
\text{`` } & m=0,\ n=0,\ t=0 & (2)\\
\text{'1'} & m=0,\ n=0,\ t=1 & (3)\\
Ad(A,m,B,n-1,Tr(\text{'0'}+B[n]))\text{✦}Cifra(\text{`0'},B[n]) & m=0,\ n>0 & (4)\\
Ad(A,m-1,B,n-1,Tr(a[m],B[n]))\text{✦}Cifra(a[m],b[n]) & m>0,\ n>0 & (5)
\end{cases}
$$

```
Program   Adunare_Numere_siruri_cifre_baza_10;   { Adun_Str[n], n<=30 }
Const     n=31;
Type      Numar = String[n];

Function Adun (a, b : Numar) : Numar;        { Adun := A+B }
  Function Ad (m, n, t : Byte) : Numar;      { Aduna primele m resp. n  cifre }
  Var c:Char;
    Function Tr (a:Char):Byte;       { Tr:=a+b[n] Div 10 ;  a = 0 sau a[m] }
    Var s : Byte;
    Begin    Tr:=0;{  c:=a+b[n] Mod 10    }
      s:= Ord(a)+Ord(b[n])-$60+t;   { s=suma cifrelor 0..19 }
      If s>9 Then Begin Tr:=1; s:=s-10 End;
      c:= Chr(s Or $30);                 { c=caracterul sumei resturilor }
    End;
  Begin
  If n=0 Then  If  t=0  Then  Ad:=''                              { Cazul (2) }
            Else       Ad:='1'                                    {       (3) }
          Else If m=0   Then Ad:=Ad(m  ,n-1,Tr( '0'))+c           {       (4) }
       Else Ad:=Ad(m-1,n-1,Tr(a[m]))+c                            {       (5) }
  End;
```

```
Begin
  If Length(a)>Length(b)
       Then Adun := Adun (b,a)          { Cazul (1),    m>n }
       Else Adun := Ad (Length(a),Length(b),0)          { Initial Transportul = 0 }
End;   {Adun }

Function  Sp(a:Numar) : Numar;                { Completeaza la stanga cu spatii }
Begin
       If Length(a)<n          Then  Sp:=Sp(' '+a)
                       Else  Sp:=a
End;

Var       a, b   : Numar;

Begin
  Write  (' Dati a : ');    Readln (a) ;
  Write  (' Dati b : ');    Readln (b) ;
  Writeln(Sp(a)+'  + ');                  {  Tipareste :       1234      +          }
  Writeln(Sp(b));                         {                     123                 }
  Write  (Sp(Adun (a,b)));                {                    1357                 }
  Readln
End.
```

Facultativ

Definire simultană (Forward) :

*Function / Procedure S*1 *(. . .) ;* ***Forward****;* *{ Antetul lui S*1 *}*

...

*Function / Procedure S*2 *(. . .) ;* *{ Subprogramul S*2 *apelează S*1 *}*
*Begin ... S*1*... End;*

*Function / Procedure S*1 *(. . .);* *{ Subprogramul S*1 *apelează S*2 *}*
*Begin ... S*2*... End;*...

" Ce functie (procedură) se va scrie prima **?** "

Ex. Pentru a compara două numere (a<b, a>b, a=b) reprezentate prin şirul (caracterelor) cifrelor, vom utiliza trei funcţii simultane (mutuale):

a<b → b>a,

a=b → *Not* a<b şi *Not* a>b,

a>b → NrCif(*a*) > NrCif(*b*) sau

NrCif(*a*) = NrCif(*b*) → Cat(*a*)>Cat(*b*) sau Cat(*a*)=Cat(*b*) şi Rest(*a*)>Rest(*b*) {/10}.

```
Program   Comparare_Numere_Str_cifre_baza_10;          { < , > , = ,  n<=10 }
Const     n=10;
Type      Numar = String[n];
Function  MaiMare(a,b : Numar) : Boolean;   Forward;
Function  MaiMic (a,b : Numar) : Boolean;
Begin     MaiMic:=MaiMare(b,a)   End;
Function  Egale (a,b : Numar) : Boolean;
Begin     Egale := Not MaiMic(a,b) And Not MaiMare(a,b) End;
Function  MaiMare (a,b:Numar) : Boolean;    Var m,n:Byte;
Begin                          m:=Length(a); n:=Length(b);
        MaiMare:=(m>n) Or                                    { |A| > |B| }
                       (m=n) And (m>0) And                   {    =      }
       (MaiMare(Copy(a,1,m-1),Copy(b,1,n-1)) Or              { Cat(A) > Cat(B) }
      (Egale  (Copy(a,1,m-1),Copy(b,1,n-1)) And (a[m]>b[n])))  { Am > Bn }
End;
Var       a,b   : String;
Begin
  Write  (' Dati a : ');    Readln (a) ;          Write  (' Dati b : ');    Readln (b) ;
  If MaiMic   (a,b) Then Write (' a < b ')   Else
  If MaiMare(a,b) Then Write (' a > b ')   Else
  If Egale    (a,b) Then Write (' a = b ')   Else                    Write (' a ? b ');  Readln
End.
```

3.2. Proceduri şi funcţii predefinite

Citirea şi scrierea datelor :

Nu exista instructiuni → Apel Proceduri

Read (<listă_variabile>)
Readln (a,b,c); // 1 2 3 xxx
Readln (v); // Simpla, Nu ***enum***,

Write (<listă_expresii>)
Writeln (a,b,c); // 1 2 3
Writeln ('…'); // …
Writeln (3.1415:5:2, 7.1:5:2, 9:3); // _3.14__7.10__9

Readln (a:5:2, b:5:2, k:3); // _3.14__7.10__9

La definirea expresiilor se folosesc şi funcţii. Tabelul următor conţine câteva funcţii predefinite (Pascal).

Functia	S e m n i f i c a t i a
abs (i)	valoarea (întreagă) absolută a întregului i
abs (x)	valoarea (reală) absolută a realului x
sqr (i)	pătratul întregului i (dacă i*i < Maxint) (valoare întreagă)
sqr (x)	pătratul numărului real x
sqrt (x)	radical din x, x>=0
int (x)	partea întreagă a numărului real x
trunc (x)	întregul obținut din numărul real x prin eliminarea părții fracționare
round (x)	cel mai apropiat întreg de numărul real x
frac (x)	valoarea x − int(x) , pentru x real
exp (x)	e la puterea x, x întreg sau real
ln (x)	logaritm natural din x, x întreg sau real>0
sin (x)	sinusul lui x, x reprezintă măsura în radiani a unui unghi
cos (x)	cosinusul lui x,
arctan (x)	arctangenta din x,
succ (i)	succesorul valorii ordinale i
pred (i)	predecesorul valorii ordinale i
ord (e)	numărul de ordine al valorii lui e în tipul expresiei ordinale e
chr (i)	caracterul de ordin i în tipul *Char*
odd (i)	funcție logică cu valoarea *True* dacă *i* este impar și *False* dacă este par

Tipul şir_caractere (String) :

- Util.: ~ ***Array Of Char***

- Decl.: ***String*** [[***m***]] Lung. Max. = ***m*** <= **255.**

Operatori pentru ***String*** :

a) Operaţia de **concatenare** a două şiruri este notată cu **+** . De exemplu 'Algoritmica,' + ' Programare ' are valoarea 'Algoritmica, Programare'.

b) Operatorii **relaţionali** permit compararea a două şiruri utilizând ordinea lexicografică (utilizată în dicţionare, cărţi de telefon, etc.) :
= şi **<>** pentru egalitatea respectiv neegalitatea a două şiruri ,
<, >, <=, >= pentru compararea lexicografică.

Ex. :
'Alb' < 'Albastru';
'Ionescu' < 'Popescu' ; '0...' < '9...' < 'A...' < 'Z...' < 'a...' < 'z...'.

Facilităţi (4F & 4P) ale tipului ***String*** (în plus faţă de *Array Of Char*) :

Valorile variabilelor şi expresiilor de tip *String* pot fi citite respectiv tipărite, ex.:
Mesaj:='Numele autorului'; Write ('Dati '+Mesaj+' : '); Readln (s);

- ***Length*** *(S)* (=Ord(S[0])) returnează lungimea şirului *S* (*Length*('mare')=4),
- ***Copy*** *(S,p,l)* returnează subşirul (lui *S*) de lungime *l* începând din pozitia *p* (Copy('Marinescu',2,4)='arin'),
- ***Concat*** *(S1,S2,...,Sn)* (=S*1*+*S2*+...+*Sn*) (*Concat*('Con','Cat')='ConCat'),
- ***Pos*** *(x,S)* det. poz. subşirului *x* în şirul *S* sau 0 dacă şirul *S* nu conţine ca subşir pe *x* (*Pos*('in','Marinescu')=4 iar *Pos*('pop','Popescu')=0),

- ***Delete*** *(s,p,l)* şterge din *s* începând din poz. *p* , *l* caractere, ex.secventa :
  s:='Ionescu'; *Delete* (s,4,4); Write (s); va tipari 'Ion';
- ***Insert*** *(x,S,p)* inserează şirul *x* în şirul *S* la poziţia *p*. De ex.:
  s:='Alg.Progr.'; *Insert*(' şi ',s,5); Write (s); va tipari 'Alg. şi Progr.';
- ***Str*** *(e,S)* depune în *S*, şirul cifrelor val. Expr.*e* , care poate avea ataşat un format (:*n*:*m* ca şi *Write*). Ex. : *v*:=5/2; Str (*v*:5:2,*S*); va depune în *S* şirul ' 2.50'.
- ***Val*** *(S,v,Cr)* examinează şirul *S*. Dacă este corect atunci va depune în *v* valoarea iar în *Cr* val. 0. Dacă şirul *S* conţine caractere nepermise, atunci în *v* se depune valoarea 0 iar în variabila *Cr* (de tip întreg) poziţia primului caracter nepermis.
  Val ('1234',v,Cr); are ca efect : v=1234 şi Cr=0 , iar
  Val ('19d7',v,Cr); are ca efect : v=0 şi Cr=3 (pe poziţia 3 se află 'd').

Problema pe care o vom rezolva utilizând tipul string este următoarea :

"Să se calculeze valoarea unei expresii aritmetice de forma : n1⊙n2[=] unde : n1,2 sunt două numere reale, ⊙ este un operator aritmetic (+, –, *, ×, / sau :) iar semnul egal la sfârşitul expresiei poate lipsi. Expresia (simplă, cu un singur operator şi fără paranteze) este dată sub forma unui şir de caractere, iar rezultatul se va afişa cu două zecimale. "

De exemplu pentru '123+45' rezultatul este 168,00 , iar pentru '123.3:3=' rezultatul este 41,10 .

Problema o vom rezolva astfel :

e - reprezintă expresia sub forma unui şir de caractere, iar *r* este rezultatul (valoarea expresiei) dată tot sub forma unui şir de caractere în care se va înlocui marca zecimală '**.**' cu '**,**' . Dacă expresia nu se termina cu caracterul '=', acesta va fi adaugat şirului introdus.

Se determină poziţia operatorului (*p*). Inseamnă că *o=e[p]* va fi caracterul ce va desemna operaţia ce urmează a fi facută. *n*1 este scris cu caracterele *e*1...*ep*-1 , deci poate fi obtinut cu procedura *Val*, iar apoi se sterg primele *p* caractere, ceea ce înseamnă că în expresia *e* mai ramane '*n*2='. În continuare, se procedează analog, extragând pe *n*2, din şirul rămas , mai puţin ultimul caracter. După tipărirea rezultatului (a şirului *r*) se va citi alt şir (altă expresie) până când se va introduce expresia vidă (şirul vid) adică se tastează doar *Enter* în momentul citirii expresiei.

Programul Pascal este următorul :

```
Program  Valoarea_Expresiei;  (* '123+22=' *)  (* n1 o n2 = ? *) (* o ∈ {+,-,x,:,*,/} *)
Var  e,r : String;    o  : Char;
      n1, n2 : Real; p , er : Integer;
Begin
 Repeat   Write (' Dati expresia  (n1 o n2 =) : '); Readln (e);   If e>'' Then Begin
     If Pos ('=',e)=0 Then e:=e+'=';
     Val (e,n1,p); Val (Copy(e,1,p-1),n1,er);              { p:=Poz(o) }   {n1:=...}
     o:=e[p];       Delete (e,1,p);     Val (Copy(e,1,Length(e)-1),n2,er);
     Case o Of
           '+' : Str (n1+n2:8:2,r);                '-' : Str (n1-n2:8:2,r);
        'x','*' : Str (n1*n2:8:2,r);             ':','/' : Str (n1/n2:8:2,r)
         Else   r:=' Operator necunoscut.'+o
     End;      p:=Pos('.',r); If p>0 Then r[p]:=',';        Writeln (r)         End
 Until e=''
End.
```

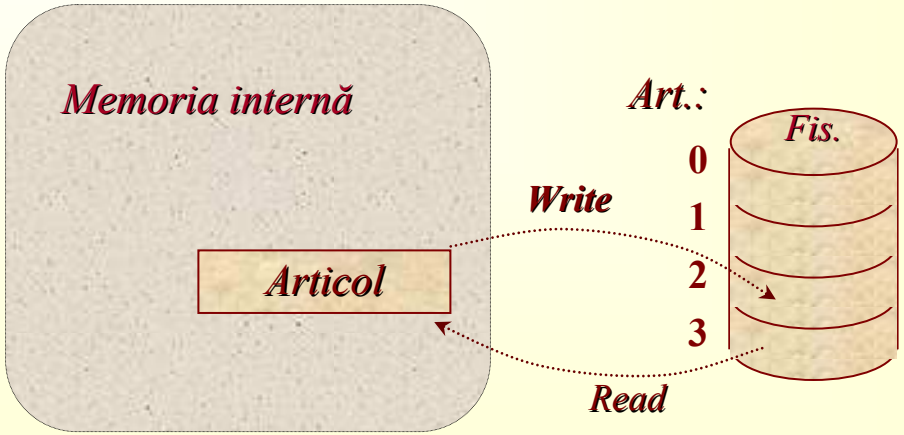
5. Fişiere text

- *Fişiere Pascal.*
- *Utilizarea fişierelor text.*
- *Fişiere cu tip*
- *Operaţii cu fişiere*
- *Fişiere fara tip – binare*

5.1 *Fişiere text ~ Tipuri de acces*

Fişierul poate fi considerat ***o colecţie de date stocate pe un suport extern*** (în general magnetic) unde pot fi păstrate atâta timp cât este necesar. Un fişier este format dintr-o succesiune de elemente (*înregistrări* sau *articole* numerotate de la 0, alcătuite la rândul lor din *câmpuri*) având toate acelaşi tip, care constituie tipul de bază al fişierului. Transferul informaţiilor între memoria internă şi cea externă (fişier) se realizează la nivel de articol. Pentru aceasta, în memoria internă va trebui să declaram o variabilă având acelaşi tip cu tipul înregistrărilor fişierului iar prin operaţiile de intrare/ieşire (***Read/Write***) se va realiza ctitirea din, respectiv scrierea în, fişier.

Accesul la componentele unui fişier poate fi *secvenţial* sau *direct*. În acces secvenţial, articolele sunt prelucrate (accesate) succesiv (unul după altul), începând cu primul pană la ultimul (Art.0, Art.1,...,Art.*n*-1).



Dacă în cazul accesului secvenţial ordinea de transfer a înregistrărilor este dată de ordinea acestora în fişier, acessul direct permite transferul în orice ordine (aleator) prin precizarea numărului de ordine din cadrul fişierului (0,1,...,*n*-1 , dacă fişierul are *n* articole).

În vederea prelucrării, fişierul va trebui **deschis** iar la terminarea **operaţii**lor fişierul va trebui **închis**. La deschiderea fişierului, indicatorul (capul de citire-scriere) este poziţionat pe prima componentă. Indicatorul se deplasează automat cu o poziţie spre sfârşit, la fiecare operaţie de scriere sau citire din fişier.

Deschiderea unui fişier (pregatirea lui pentru lucru) se face în doua etape:
- În primul rând va trebui să precizăm ce fişier se deschide (se face asocierea între *variabila* fişier şi *specificatorul* de fişier, corespunzator fişierului dorit) utilizând procedura ***Assign (Variabila_Fişier, Specificator_Fişier)***. Această asociere fiind facută, de acum înainte vom utiliza doar *Variabila_Fişier* pentru a referi fişierul de lucru (pe care o vom nota în continuare mai simplu cu *F*).
- Deschiderea propriuzisă se va efectua prin apelul procedurii ***Rewrite(F)*** sau ***Reset(F)*** după cum fişierul care va fi deschis urmează să se creeze sau să se consulte .

Operaţiile de intrare / ieşire sunt:

Read(F,Art) respectiv

Write(F,Art)

unde am notat prin *Art* variabila din memoria internă (declarată în program) în , sau din care se face transferul din/în fişier.

După terminarea operaţiilor de transfer, fişierul poate fi închis prin:

Close(F).

O functie utilă în programele care lucrează cu fişiere este funcţia booleană

Eof(F)

care ne spune dacă s-a ajuns la sfârşitul fişierului (dacă a fost detectată marca de sfârşit de fişier).

În programele Pascal este posibilă ştergerea unui fişier apelând procedura:

Erase(F).

De asemenea, este posibilă schimbarea numelui unui fişier în timpul execuţiei unui program. Această redenumire se poate realiza prin apelul:

Rename(F, Noul_Specificator_de_Fişier) .

5.2. *Proceduri şi funcţii predefinite pentru fişiere text*

Conţinutul unui ***fişier text*** poate fi înţeles ca un text dintr-o carte, organizat pe un număr de linii compuse din caractere (rânduri de tip *string*). Într-un fişier *text* sfârşitul fiecărei linii este marcat prin două caractere speciale ***CR*** (de la *Carriage Return*) şi ***LF*** (de la *Line Feed*). Avantajul folosirii acestor fişiere constă în faptul că ele se pot vizualiza din *Norton Commander* sau *Turbo Pascal* şi chiar pot fi corectate folosind funcţia de *editare* a acestora.

Declararea unui fişier text se realizează simplu prin scrierea cuvântului cheie *Text.*

De exemplu :

Var f, g : ***Text****;*

Citirea datelor dintr-un fişier text se poate efectua prin apelul procedurii :

Read (F, Listă_Var_Intrare); sau

Readln(F, Listă_Var_Intrare);

transferul făcându-se din fişierul *F* , în aceeaşi manieră ca şi de la tastaură.

Scrierea datelor într-un fişier text se poate efectua prin apelul procedurii :

Write(F,Listă_Expresii_Ieşire); sau

Writeln(F Listă_Expr._Ieşire);

scrierea făcându-se în fişierul *F* ca şi pe ecran.

Un prim exemplu îl constituie o procedură care citeşte o matrice (dimensiunile şi valorile acesteia) dintr-un fişier text.

Considerăm că aceste date sunt organizate în fişierul 'Matrice.Txt' astfel :

- primul rând conţine numărul de linii şi numărul de coloane (*m* şi *n*),
- următoarele *m* rânduri conţin valorile corespunzatoare fiecărei linii ale matricei.

```
Procedure  Cit_Mat  (Var A :matrice;  Var m,n : Integer);
Var   i, j : Integer;      F : Text;
Begin
        Assign  (F,'Matrice.Txt');  Reset (F);
        Readln (F,m,n);
        For i:=1 To m Do
            For j:= 1 To n Do
                Read (F,A[i,j]);
        Close (F)
End;
```

```
3 4
1 2 3 4
5 6 7 8
9 0 1 2
```

Următorul exemplu este un program Pascal care traduce un text aflat într-un fişier conform unui dicţionar (care se află stocat într-un fişier text având numele 'Dictio.Txt').

Dicţionarul este compus dintr-un şir de perechi de cuvinte de forma (*St,Dr*) şi se construieşte citind câte un rând de forma *Stânga=Dreapta* din fişierul text.

De exemplu, conţinutul fişierului 'Dictio.Txt', tipărit în continuare, poate "traduce" un Program Pascal in ***Pseudocod***.

Pentru a realiza o traducere, se citeste câte un rând din fişierul iniţial, se traduce conform dicţionarului, după care se depune (rândul tradus) în fişierul final (tradus).

Traducerea unui rând se execută astfel: se ia fiecare cuvânt din dicţionar (aflat în partea stangă) şi se verifică dacă acesta se află în rândul curent. Dacă da, atunci se va înlocui în rând cuvântul găsit cu perechea acestuia (partea dreaptă din dictionar).

Program=Algoritmul
Readln=Date
Read=Date
Writeln=Rezultate
Write=Rezultate
For=Pentru
Do=Executa
Repeat=Repeta
Until=Pana_Cand
While=Cat_Timp
End.=Sfarsit.
. . .

Programul Pascal este următorul :

```
Program  Traducere_Text;
Const    Mc = 100;
Type     Sc = String[10];        Rd = Record  St, Dr : Sc End;              {   (St,Dr)  }
Var      Fis, Tra, Dic : Text;    Numei,Numef : String[20];
         Dict : Record Nrc : Byte;            { Numarul de cuvinte din dictionar }
                    Cuv : Array [1..Mc] Of Rd
                End;
        Rând : String;{ un rând din dictionar  de forma  St = Dr  }
        p,i : Integer;   { p memoreaza pozitia semnului = în Rând }
Begin      {  ( delimiteaza partea stanga de partea drepta) }
   Assign (Dic,'Dictio.Txt');  Reset (Dic);            { Deschide Dictionarul }
   With Dict Do Begin          { Constr. Dictionarul în memorie în variabila Dict.}
            Nrc:=0;
      While Not Eof(Dic) Do Begin
         Readln (Dic,Rând); p:=Pos('=',Rând);      { Cuv[Nrc] → Dict }
         Nrc:=Nrc+1;
         With Cuv[Nrc] Do Begin
            St:=Copy(Rând,1,p-1);          { St := ...  p    }
            Dr:=Copy(Rând,p+1,Length(Rând)-p)   { Dr :=     p ... }
         End
      End
    End;
```

```
Write (' Numele fişierului de tradus : '); Readln (Numei);              { Fiş. de tradus }
Write (' Numele fişierului rezultat  : '); Readln (Numef); { Fiş. tradus }
Assign(Fis, Numei); Reset  (Fis);      { Deschide fişierele }
Assign(Tra, Numef); ReWrite(Tra);
While Not Eof(Fis) Do Begin
  Readln (Fis,Rând);
  With Dict Do   { Traduce Rând }
    For i:=1 To Nrc Do  With Cuv[i] Do
      Repeat
        p := Pos (St,Rând);
        If p>0 Then Begin            { Exista cuvantul St în Rând ? }
          Delete (Rând,p,Length(St));          { Inlocuieste St cu Dr }
          Insert (Dr,Rând,p)
        End
      Until p = 0;
    Writeln(Tra,Rând)       { Scrie Rând-ul tradus }
  End;
Close (Fis); Close (Tra)       { Inchide fişierele }
End.
```

Facultativ

5.3. *Operaţii cu fişiere*

Principalele operaţii asupra fişierelor sunt :

- ***Creere*** (generarea unui fişier, care se execută o singură dată, când ia fiinţă fişierul),
- ***Actualizare*** (corectarea fişierului, punerea la punct în cazul în care datele nu sunt corecte sau nu mai sunt actuale), şi
- ***Listare*** (etapa finală de obţinere a rezulatatelor, a situaţiilor finale).

Actualizarea fişierelor poate fi efectuată *secvenţial* (articol după articol) sau *direct* (aleator, precizând numărul de ordine al unui articol). În operaţia de actualizare se poate efectua :

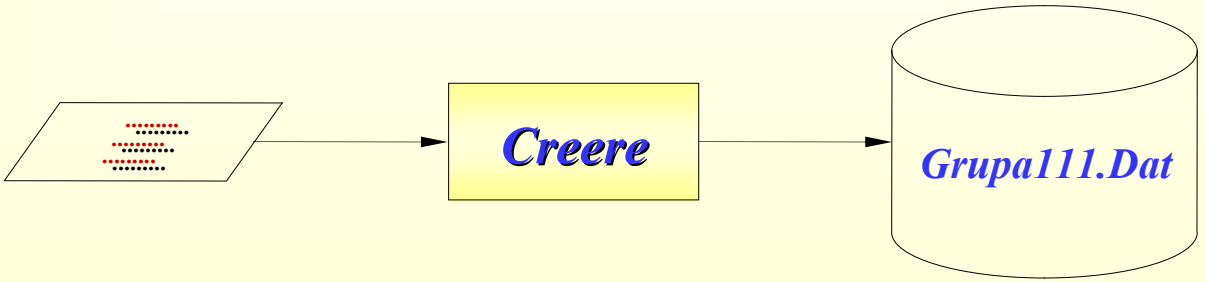
- ***Adăugarea*** de noi articole,
- ***Modificarea*** unor articole (prin modificări de câmpuri ale acestora), sau
- ***Stergerea*** de articole.

Operaţiile descrise anterior vor fi exemplificate în cele ce urmează prin programe Pascal.

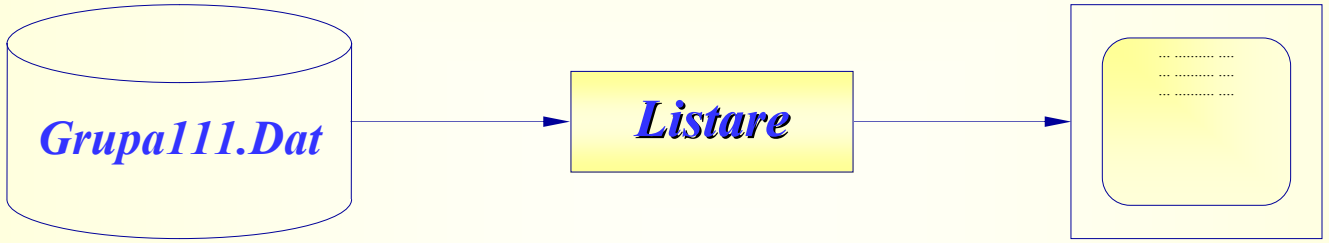
În exemplul următor se doreşte *creerea* fişierului ‘Grupa111.Dat’, cu articole de forma:

Nr_Leg	*Nume*	*Medie*
Word	*String[25]*	*Real*

Pentru a crea acest fişier, se citeşte de la tastatură câte un articol (care conţine informaţiile despre un student) apoi acesta se scrie în fişier.

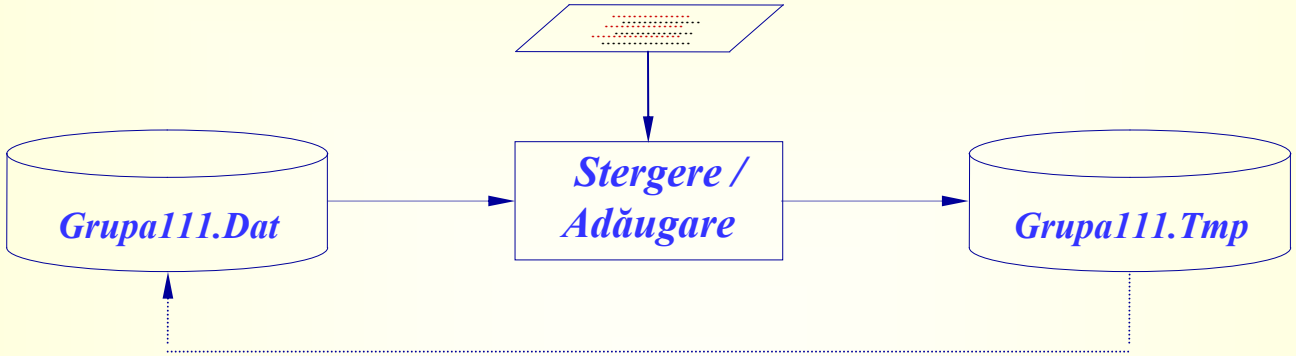


Următorul program realizează ***listarea*** (tipărirea articolelor) fişierului *Grupa111.Dat* pe ecran. Pentru a tipări acest fişier, se citeşte din acesta câte un articol, apoi acesta se afişează pe ecran :



Următoarele exemple sunt dedicate ***actualizării*** fişierului 'Grupa111.Dat'. Pentru aceasta, se citesc de la tastatură datele (care conţin informaţiile necesare actualizării) conform cărora se corectează fişierul.

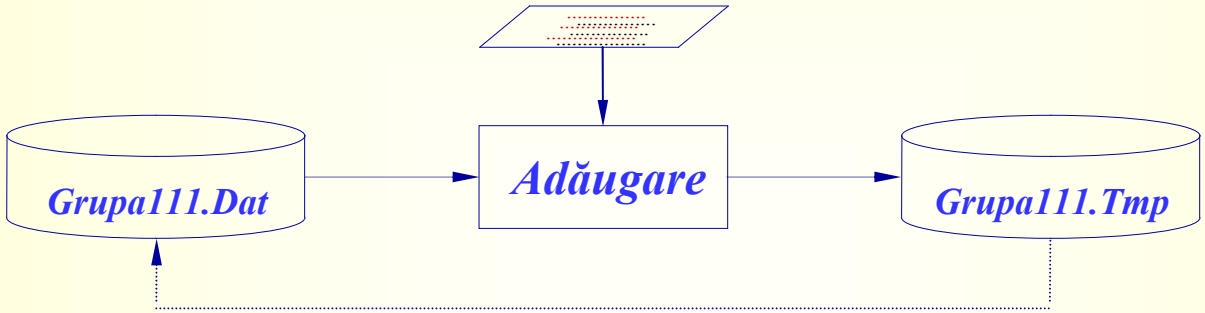
Atât pentru ***ştergerea*** de articole dintr-un fişier cât şi pentru ***adăugarea*** de noi articole este necasară utilizarea unui fişier temporar, aşa cum se poate vedea în figura următoare:



Dacă se doreşte ştergerea aleatoare a unor articole din fişier, numărul curent al acestor articole va fi introdus de la tastatură. Aceste articole vor fi omise la copierea articolelor din fişierul iniţial (‘Grupa111.Dat’) în fişierul temporar (‘Grupa111.Tmp’).

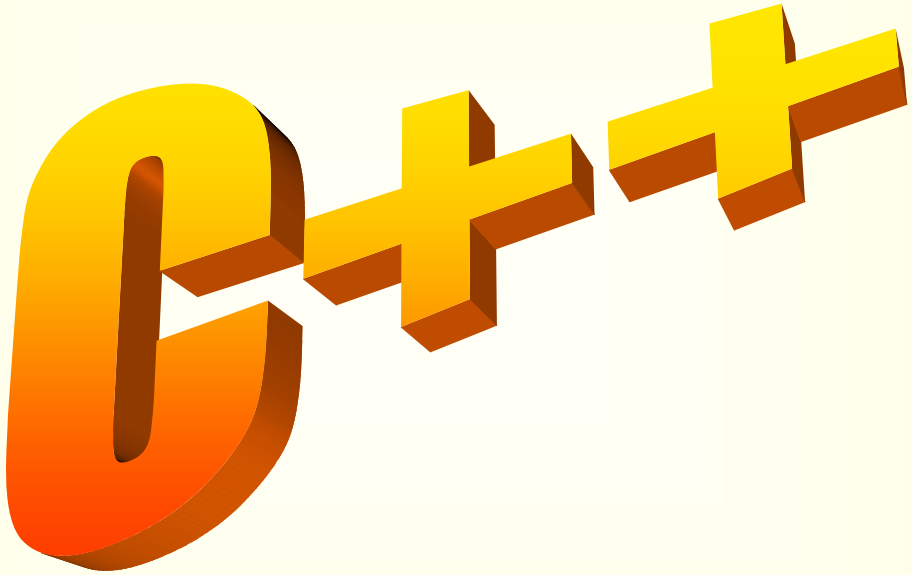
Pentru ca fişierul actualizat să poarte acelaşi nume iniţial, se va proceda astfel: fişierul iniţial se şterge (*Erase(Fis_Vechi)*) iar fişierul final se va redenumi (*Rename(Fis_Temporar,Nume_Fişier))* aşa cum se poate vedea în exemplul următor.

Programul pentru ***adăugarea*** de noi articole în fişier este prezentat în continuare şi lucrează în felul următor : citeşte de la tastatură câte un articol pe care îl adaugă *înaintea* articolului cu numărul de ordine *Nr_Crt* (introdus tot de la tastatură).



Toate articolele până la cel precizat se copiază din fişierul iniţial în cel temporar, apoi se scrie noul articol (citit anterior). Se citeşte din nou alt articol care va fi adăugat şi aşa mai departe până când nu mai sunt articole de adăugat. Restul articolelor care au mai rămas în fişierul iniţial se vor copia şi ele în fişierul final.

Aşa cum s-a procedat şi la *Stergere,* se vor depune datele obţinute în fişierul iniţial (apelând ***Erase***... şi ***Rename***...) .


C++

1.3. Funcţii

O **funcţie** este formată dintr-un ***antet*** şi un ***bloc*** (***corp***). Ea poate fi apelată dacă a fost definită în întregime sau doar antetul său.

Antetul unei funcţii are următorul format:

Tip Nume (Listă_parametri_formali)

unde:

- ***Tip*** este tipul valorilor funcţiei (codomeniul);
- ***Nume*** este un identificator (literă urmată eventual de litere sau cifre);
- ***Listă_parametri_formali*** conţine *parametrii formali* separaţi prin ‘,’.

Exemplu:

```
int Min (int a, int b)
{       if  (a<b)  return a;  else  return b; }
```

Obs. ***Prototipul*** unei funcţii este ***antetul*** acesteia urmat de ‘;’ .

Corpul unei funcţii are următoarea structură:

```
{
        Declaraţii
        Instrucţiuni
}
```

Exemple:

```
int Cmmdc (int a, int b)                          //  Cmmdc(a,b)
{
    if (b= =0) return a;
          else   return   Cmmdc(b,a % b);     //  Cmmdc(b,a  Mod  b);
}
int cmmdc (int a, int b)                          //  cmmdc(a,b)
{   int    rest;
    do {rest=a%b;
          a=b;
          b=rest;   }
    while (rest!=0);                  //  rest ≠  0;    sau    while (rest) ;
    return a;
}
```

4. Operaţii de intrare/ieşire

4. Operaţii de *intrare/ieşire*

În limbajul C nu există instrucţiuni de ***intrare/ieşire***, aceste operaţii ralizându-se prin ***funcţii*** aflate în bibliotecile standard.

Implicit, unui program i se ataşează fişierele:

- ***stdin*** (intrare standard),
- ***stdout*** (ieşire standard),
- ***stderr*** (ieşire standard pentru erori),
- ***stprn*** (ieşire pentru imprimantă),
- ***stdoux*** (intrare/ieşire serială).

4.1. Funcţia *Printf*

Această funcţie realizează afişarea după un şablon, având următorul format:

int ***printf*** (*Control* [, *Listă_Expresii*]);

unde *Control* este *şablonul* (*formatul*) scris sub forma unui şir de caractere care conţine *mesaje* şi *specificatori de format* corespunzători valorile expresiilor din listă. *Un specificator de format* se descrie astfel :

% [–] [*m*[.*n*]] [l] [*f*] .

unde:

- [–] - determină alinierea la stânga, implicită fiind la dreapta,
- [*m*] - precizează lungimea câmpului,
- [.*n*] - precizează lungimea părţii zecimale, respectiv numărul de caractere,
- [l] - conversia se va efectua din format intern *long*,
- [*f*] - determină tipul conversiei.

… 4.1. Funcţia *Printf*

Tipul conversiei [*f*] este precizat prin unul din următoarele caractere:

- *d* - *int* → zecimal extern,
- *o* - *int* → octal extern,
- *x* - *int* → hexa extern (0...9,a...f),
- *X* - *int* → Hexa extern (0...9,A...F),
- *u* - *unsigned* → zecimal extern (fara semn),
- *c* - *binar intern* → caracter (*char*),
- *s* - *string* (sir de coduri ASCII terminat cu \0=NUL) → sir de caractere,
- *f* - *float* sau *double* → zecimal extern [*m*[.*n*]], implicit *n*=6,
- *e* - *float* sau *double* → zecimal extern forma exponentiala ($b*10^e$),
- *E* - *float* sau *double* → zecimal extern forma exponentiala ($b*10^E$),
- *g* - se alege dintre variantele *f* sau *e* reprezentarea minima,
- *G* - se alege dintre variantele *f* sau *E* reprezentarea minima.

… 4.1. Funcţia *Printf*

Funcţia ***printf*** returnează numărul de octeţi afişaţi dacă operaţia a decurs corect, iar în caz contrar -1 (EOF):

☞ if (EOF == ***printf*** (*Control , Lista_Expresii*)) … *eroare* … ;

Exemple:

```
short Zi=1; char Luna[]="Ianuarie"; unsigned An=2003;
float Ina=1.8;
printf(" Zi:%d, Luna:%3.3s., An:%u \n",Zi,Luna,An);      // Zi:1, Luna:Ian., An:2003
printf(" Inaltime(m):%4.2f \n",Ina);                     // Inaltime(m):1.80
```

4.2. Funcţia *Scanf*

Aceasta functie realizeaza citirea datelor dupa un sablon, având urmatorul format:

int ***scanf*** (*Control* , *Lista_**Adrese**_de_Variabile*);

unde *Control* este sablonul (formatul) scris sub forma unui sir de caractere care contine eventual *texte (*obligatorii la intrare*)* si *specificatori de format* corespunzatori tipurilor variabilelor din lista. Specificatorii de format sunt asemanatori celor prezentati la functia ***printf***, realizând însa conversiile inverse:

% [*] [*m*] [l] [*f*] ,

unde:

- [*] - un caracter optional,
- [*m*] - precizeaza lungimea maxima a câmpului,
- [l] - conversia se va efectua din format intern *long*,
- [*f*] - determina tipul conversiei.

… 4.2. Funcţia *Scanf*

Tipul conversiei [*f*] este precizat prin unul din următoarele caractere:

- *d* - *int* ← zecimal extern,
- *o* - *int* ← octal extern,
- *x* - *int* ← hexa extern (0...9,a...f),
- *X* - *int* ← Hexa extern (0...9,A...F),
- *u* - *unsigned* ← zecimal extern (fara semn),
- *c* - *binar intern* ← caracter *(char),*
- *s* - *string* ← sir de caractere terminat la spatiu sau dimensiunea *m,*
- *f* - *float* ← flotant extern.

… 4.2. Funcţia *Scanf*

Adresele variabilelor de intrare sunt date prin operatorul de adrese **&** plasat înaintea identificatorului fiecarei variabile (simple!):

[&] *Variabila* (nu este necesar pentru tablouri).

Exemplu:

```
short Zi;  char Luna[13];  unsigned An;  float Ina;
scanf(" %d %s %u %f ", &Zi,   Luna, &An, &Ina);          // 1 Ianuarie 2003 1.80
```

Functia ***scanf*** returneaza numarul de câmpuri citite corect. Sfârsitul de fisier (Ctrl/Z) poate fi verificat prin valoarea returnata EOF:

if (EOF == ***scanf*** (*Control , Lista_Expresii*)) … *Sfârsit* … ;

Exemplu:

```
if (EOF==scanf(" %d %s %u %f", &Zi,   Luna, &An, &Ina))  printf("Ctrl/Z");
   else { printf(" Zi:%d, Luna:%3.3s., An:%u \n",Zi,Luna,An);  printf(" Inaltime(m):%4.2f \n",Ina);  }
```

4.3. Funcţia *PutChar*

Această funcţie realizează tipărirea unui caracter al cărui cod ASCII este precizat printr-o expresie :

putchar (*Expresie*);

4.4. Funcţia *GetChar*

Aceasta, returnează codul ASCII al caracterului citit (pentru Ctrl/Z → EOF=-1):

getchar ( );

Exemplu:

```
char   c;
do     putchar (((c=getchar())>'Z')? c^' ' : c);      // Litere mici → LITERE MARI
while (c!='.');
…
```

Litere mici in Litere MARI ! ↵ **→ getchar**
LITERE MICI IN LITERE MARI ! **putchar ←**
Se termina la . (Punct) ↵ **→ getchar**
SE TERMINA LA . **putchar ←**

4.5. Funcţia *GetChe*

Funcţia returnează codul caracterului citit *direct* şi îl afişează (în *ecou*):

int ***getche*** ( );

Exemplu:

```
do  putchar (((c=getche())>'Z')? c^' ' : c);  while (c!='.');   // se termina cu .
```

LLiltTeErReE mMilcCil iInN LLiltTeErReE MMAARRII..

4.6. Funcţia *GetCh*

Funcţia returnează codul caracterului citit *direct* fără a mai fi afişat:

int ***getch*** ( );

Exemplu:

```
int ReadKey()
{       int car=getch();
        if (car) return car;
        else return getch();  //  a fost apăsată o tastă functională (0,cod ASCII)
}
```

4.7. Funcţia *KbHit*

Această funcţie returnează starea tastaturii (a fost apăsată o tastă ?):

int ***kbhit*** ( );

Exemplu:

```
# include <conio.h>
# include <stdlib.h>
# include <stdio.h>
# include <iostream.h>
void main ()
{  char   c[8],i=-1;
   cout << " Password : ";
   do {while (!kbhit()) { putchar(random(27)+'A'); putchar('\b'); }    // kbhit()=KeyPressed
         if ((c[++i]=getch())!='\r') putchar('*'); else putchar(' ');    }
   while  (c[i]!='\r'); c[i]=0;
   cout << " Password = " << c;            getch();
}
```

4.8. Funcţiile *PutS, GetS*

Acestea citesc (inclusiv spaţiile şi taburile până la *Enter*), respectiv tipăresc un string:

```
char *gets (char *s);
int  puts (const char *s); .
```

Exemplu:

```
#include <stdio.h>
#include <conio.h>
#include <iomanip.h>
#include <iostream.h>
void  main ()
{                                                          clrscr();
   puts(" Dati o propozitie ...  <Cr> ");
   char s[100];  gets (s);
   int i=0;   do cout <<hex<<setw(3)<<int(s[i]); while (s[++i]);      getche();
}
```

```
Dati o propozitie ...  <Cr>
 I/E    pt. str.
 20 49 2f 45  9 70 74 2e 20 73 74 72 2e
```

4.9. *Stream-uri*

Dispozitivelor standard de intrare-iesire li s-au atasat *streamurile* (flux de date) standard ***cin*** (pentru *stdin*) si respectiv ***cout*** (pentru *stdout*), care permit efectuarea operatiilor de intrare-iesire aplicând operatorul >> *streamului* ***cin***, respectiv << *streamului* ***cout***. Ierarhiile necesare sunt în fisierul ***iostream.h*** .

Exemplu:

```
#include <iostream.h>
#include <stdio.h>;
#include <conio.h>;
void main (void)
{ int i;       cout << " Dati i : "; cin >> i;
               cout << " Val. i = " << i+1 << endl; // endl=<Cr>
  char s[10]; cout << " Dati s : "; cin >> s;
               cout << " Sir. s = " << s << endl;     getch();
}
```

6. Apelul unei funcţii

Instrucţiunea de apel a unei funcţii este un caz particular al instrucţiunii expresie:

Nume_funcţie (Listă_parametri_actuali);

O funcţie poate fi apelată şi în cadrul unei expresii dintr-o instrucţiune:

... Nume_funcţie (Listă_parametri_ actuali) ... ;

… 6. Apelul unei funcţii

O funcţie poate fi utilizată doar dacă a fost definită, sau cel puţin a fost declarat ***prototipul*** (***antet*** **;**) ei într-una din următoarele moduri:

a) Tip_funcţie Nume_funcţie (Lista_parametrilor_formali);

b) Tip_funcţie Nume_funcţie (Lista_tipurilor_parametrilor_formali);

c) Tip_funcţie Nume_funcţie (void); // nu sunt parametri formali

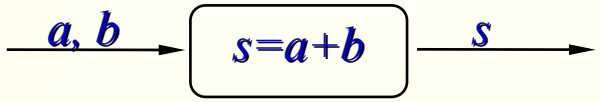
d) Tip_funcţie Nume_funcţie ( ); // nu se fac verificările de tip

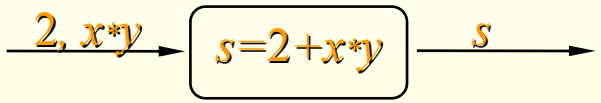
… 6. Apelul unei funcţii

Apelul implicit pentru *variabile simple* (*de bază*) este ***prin valoare***, iar pentru *tablouri* ***prin referinţă***. ***Apelul prin referinţă*** se obţine prin intermediul variabilelor de tip ***pointer*** sau a operatorului de adresă ***&***.

Parametri de intrare → *Expresii*

Parametri de iesire → *Variabile*






```
void suma (int a, int b, int& s)
{
      s=a+b;
}
```

```
{     …
      suma (2, x*y, s);
      …
}
```

… 6. Apelul unei funcţii

Revenirea dintr-o funcţie se poate realiza fie prin instrucţiunea ***return***, fie automat după ultima instrucţiune a funcţiei (situaţie în care nu se returnează nici o valoare):

return [*expresie*] ;

fiind returnată valoarea ***expresiei*** (dacă există).


```
void citeste (int& a)
{
      cin >> a;
}
```

```
int citeste (void)
{     int a;
      cin >> a;  return a;
}
```

```
void tipareste (int a)
{
      cout << a;
}
```

Exemplul 1:

```
#include <graphics.h>   #include <math.h>
int    u1,v1, u2,v2;       float  a, b,  c, d ;
int  u (float x)   { return ((x-a)/(b-a)*(u2-u1)+u1); }
int  v (float y)   { return ((y-d)/(c-d)*(v2-v1)+v1); }
void InitGraf(void)         { int Gd = DETECT, Gm;  initgraph(&Gd, &Gm, "c:\\Bc\\Bgi"); }
void ViewPort(int x1,int y1,int x2,int y2)              {u1=x1; v1=y1;
                                                u2=x2; v2=y2;   /*rectangle(u1,v1,u2,v2);*/   }
void Window(float x1,float y1,float x2,float y2)      { a=x1; d=y1; b=x2; c=y2; }
void Rectangle(float x1,float y1,float x2,float y2)    { rectangle(u(x1),v(y1),u(x2),v(y2)); }
void Bar(float x1,float y1,float x2,float y2)           { bar(u(x1),v(y1),u(x2),v(y2)); }
void Linie(float x1,float y1,float x2,float y2)         { line(u(x1),v(y1),u(x2),v(y2)); }
void Muta(float x,float y)                    { moveto(u(x),v(y)); }
void Trag(float x,float y)                    { lineto(u(x),v(y)); }
void Rot(float &x,float &y, float x0, float y0, float Alfa)  {      float xp;
      xp=(x-x0)*cos(Alfa)-(y-y0)*sin(Alfa)+x0;
      y =(x-x0)*sin(Alfa)+(y-y0)*cos(Alfa)+y0;
      x = xp;
}
```

Exemplul 2:

```
#include <iostream.h>;
#include <conio.h>;
int Sf (int& f, int k)
{ int p=0;
  while (!(f%k)) { f/=k; p++; }
  return p;
}
main ()  {                                                clrscr();
      int    n;    int f2=0; int Uc=1;
      cout << " n : ";  cin >> n;
      for (int i=2; i<=n; i++)     { int f=i;
            f2+=Sf(f,2); f2-=Sf(f,5); Uc=Uc*f%10; }
      cout << " Uc= " << Uc*((f2&=3,int(f2?f2*1.4:3))<<1)%10;
      getch();
}
```

Exemplul 3: // Calc. A∪B, A∩B \\

```
#include <iostream.h>;     #include <conio.h>;
int Card(int A[])  {    return A[0];  }
int Apart (int x, int A[])
{   for (int i=1; i<=Card(A); i++) if (x==A[i]) return 1; return 0; }
void n (int A[], int B[], int C[])
{ C[0]=0; for (int i=1; i<=Card(A); i++) if (Apart(A[i],B)) C[++C[0]]=A[i];  }
void u (int A[], int B[], int C[])
{ int  i;  for (i=0; i<=Card(B); i++) C[i]=B[i];
  for (i=1; i<=Card(A); i++) if (!Apart(A[i],B)) C[++C[0]]=A[i];  }
void Tip (char *Mult, int A[])
{  int i; cout << Mult << '{' ;
   for (i=1; i<=Card(A); i++) cout << A[i] << ",";   cout << "\b}" << endl;  }
void main (void)
{                                                          clrscr();
  int A[]={5, 1,3,5,7,9};      Tip ("  A  : ",A);
  int B []={5, 1,2,3,4,5};     Tip ("  B  : ",B);
  int AuB[10];  u (A,B,AuB);   Tip (" AuB = ",AuB);
  int AnB[10];  n (A,B,AnB);   Tip (" AnB = ",AnB);          getche();
}
```

6.1. Operatorul de adresă (&)

Acest operator (&) se poate utiliza şi pentru a defini un tip *referinţă* printr-o declaraţie de forma *tip* & (asemănător cu o construcţie de forma *tip* *, pentru *pointer*).

Cu ajutorul acestui operator putem:

- redenumi o variabilă,
- **realiza un apel prin referinţă,**
- să declarăm o variabilă de referinţă astfel:

```
tip &  parametru_formal      // par. ref. (adresă)
```

```
tip &  nume_var_ref ;        // var. de tip referinţă
```

Exemplul 4:

// Apel prin Referinţă

```
#include <iostream.h>;
void suma (int x, int y,  int *  z)   { *z  = ++x * ++y; }          // x,y → z
void Suma (int x,i nt y,  int &z)   {   z  = ++x * ++y; }          // x,y → z

void main (void)
{
  int x,y, z;
  cout << " Dati x,y : ";   cin  >> x >> y;
  suma(x,y,&z);
                    cout << "(x+1)*(y+1)=" << z << endl;
  Suma(x,y, z);
                    cout << "(x+1)*(y+1)=" << z << endl;      // mai simplu!
}
```

7. Functii standard

Macrouri de clasificare (***ctype.h***):

int ***isascii*** (*int* car);	car ∈ [0,127] ?
int ***isalpha*** (*int* car);	car este codul unui caracter alfanumeric ?
int ***isalnum*** (*int* car);	car este codul unei litere ?
int ***isupper*** (*int* car);	car este codul unei litere mari ?
int ***islower*** (*int* car);	car este codul unei litere mici ?
int ***isdigit*** (*int* car);	car este codul unei cifre zecimale ?
int ***isxdigit*** (*int* car);	car este codul unei cifre hexa ?
int ***isgraph*** (*int* car);	car este codul unui caracter afisabil ?
int ***isprint*** (*int* car);	car este codul unui caracter imprimabil ?
int ***isspace*** (*int* car);	car este spatiu, tab, Cr, Lf, Vt sau Np ?

Macrouri de transformare a simbolurilor (***ctype.h***):

int ***toascii***(*int* car);	car → [0,127] (returneazt ultimii 7 biti)
int ***toupper*** (*int* car);	car → litera mare (transforma din l în L)
int ***tolower*** (*int* car);	car → litera mica (transforma din L în l)

Conversii (***stdlib.h***):

Format intern ← Format (zecimal) extern	
int ***atoi***(*const char* *ptr);	*binar (int) ← zecimal extern*
long ***atol***(*const char* *ptr);	*binar (long) ← zecimal extern*
double ***atof***(*const char* *ptr);	*flotant (dubla precizie) ← zecimal extern*
Format extern ← Format intern	
char ****itoa*** (*int* v,*char* *s, *int* b);	s ← v_b *(val. v de tip int*, scrisa *în baza b)*
char ****ltoa*** (*long* v,*char* *s, *int* b);	s ← v_b *(val. v de tip long* scrisa *în baza b)*

Prelucrarea sirurilor de caractere (***String.h***):

char * ***str[n]cpy*** (*char* *destinatie, *const char* *sursa [, *unsigned* n]) ;

char * ***str[n]cat*** (*char* *destinatie, *const char* *sursa [, *unsigned* n]) ;

int ***str[n][i]cmp*** (*char* *sir1, *const char* *sir2 [, *unsigned* n]) ;

unsigned ***strlen*** (*char* *sir) ;

*char** ***strrev*** *(char* *sir*)* ;

*char** ***strstr*** (*const char* *sir, *const char* *subsir) ;

*char** ***strtok*** *(char* *sir, *const char* *subsir*)* ;

*char** ***strchr*** *(const char* *sir, *int* car*)* ;

*char** ***strset*** *(char* *sir, *int* car*)* ;

Functii de calcul (***math.h***):

Prototip	***Semnif.***
double ***sin*** (*double x*);	*sin(x)*
double ***cos*** (*double x*);	*cos(x)*
double ***asin*** (*double x*);	*Arcsin(x)*
double ***acos*** (*double x*);	*Arccos(x)*
double ***atan*** (*double x*);	*arctg(x)*
double ***sinh*** (*double x*);	*sh(x)*
double ***cosh*** (*double x*);	*ch(x)*
double ***tanh*** (*double x*);	*th(x)*
double ***sqrt*** (*double x*);	$\sqrt{x}$
double ***exp*** (*double x*);	e^x

double ***log*** (*double x*);	*ln(x)*
double ***log10*** (*double x*);	*lg(x)*
double ***ceil*** (*double x*);	$[x]$
double ***floor*** (*double x*);	*trunc*(x)
double ***fabs*** (*double x*);	$\|x\|$
int ***abs*** (*int x*); *	$\|x\|$
long ***labs*** (*long x*); *	$\|x\|$
double ***atan2*** (*double y,double x*);	*arctg(y/x)*
double ***pow*** (*double x,double y*);	x^y
double ***cabs*** (*struct complex z*);	$\|z\|$
double ***poly*** (*double x,int n,double a*[]);	*P(x)*

Functii pentru controlul proceselor (***process.h***):

void ***abort*** (*void*);	*termina un program la eroare*
void ***exit*** (*int cod_retur*);	*termina un program cu un cod de retur*
int ***system*** (*const char *comanda*);	*executa o c-da si ret. cod_retur* (0=*Ok*).

Functii pentru timp (***dos.h***):

```
struct date  {
        int da_year;
        int da_day;
        int da_mon;
        }
```

```
struct time  {
        unsigned char ti_min;
        unsigned char ti_hour;
        unsigned char ti_hund;
        unsigned char ti_sec;   }
```

Citeste Data curenta :	*void* ***getdate*** (*struct date *Data*);
Modifica Data curenta :	*void* ***setdate*** (*const struct date *Data*);
Citeate Ora Exacta :	*void* ***gettime*** (*struct time *OraExacta*);
Modifica Ora Exacta :	*void* ***settime*** (*const struct time *OraExacta*);

Ecranul în mod grafic (***graphics.h***):

void far ***initgraph*** (*int far* *graphdriver, *int far* *graphmode, *char far* *path) ;

void ***closegraph*** (*void*) ;

void far ***setbkcolor*** (*int* culf) ;

int far ***getbkcolor*** (*void*) ;

void far ***setcolor*** (*int* culs) ;

int far ***getcolor*** (*void*) ;

Numar de pixeli (Oriz./Vert.)	***getmaxx*** **();** ***getmaxy*** **();**
Coordonatele LPR (Ult. Pct Ref.)	***getx*** **();** ***gety*** **();**
Muta LPR (Abs./Rel.)	***moveto*** **(*x*,*y*);** ***moverel*** **(*dx*,*dy*);**
Traseaza segment din LPR	***lineto*** **(*x*,*y*);** ***linerel*** **(*dx*,*dy*);**
Traseaza segment	***line*** **(x_1,y_1, x_2,y_2);**
Deseneaza dreptunghi	***rectangle*** **(x_1,y_1, x_2,y_2);**
Deseneaza cerc	***circle*** **(*x*,*y*,*r*);**
Scrie mesaj **[** *din LPR* **]**	***outtext*****[*xy*]** **([*x*,*y*,]** ***mesaj*****);**
Pagina activa / Pagina vizuala	***setactivepage*** **(*pag*);** ***setvisualpage*** **(*pag*);**

8. Utilizarea fisierelor

Prelucrarea fisierelor se poate efectua la doua nivele:

- *Nivelul inferior* - face apel direct la sistemul de operare;
- *Nivelul superior* - utilizeaza proceduri speciale
  pentru operatii de intrare/iesire.

a) Nivelul inferior

Deschiderea unui fisier (*Open*/*Creat*) :

int **open** (*const char* * **cale**, *int* **acces**) ;

unde:

cale - este specificatorul de fisier,

acces - poate fi o combinatie (operatorul '|') a urmatoarelor valori:
O_RDONLY, O_WRONLY sau O_CREAT (scriere - creare),
O_RDWR (citire/scriere), O_APPEND, O_BINARY, O_TEXT .

```
...
int   Lun;
      Lun = open ("Fisier.Dat",O_RDONLY);
...
```

Citirea dintr-un fisier (*Read*) :

```
int read ( int Lun, void *buffer, unsigned lung ) ;
```

Scrierea într-un fisier (*Write*) :

```
int write ( int Lun, void *buffer, unsigned lung ) ;
```

Pozitionarea într-un fisier (*LSeek*) :

```
long lseek ( int Lun, long depl., int orig.) ; // O=0,1,2
```

Închiderea fis. (*Close*) :

```
int close ( int Lun) ;
```

Stergerea fis. (*UnLink*) :

```
int unlink (const char * cale) ;
```

```
                                        //  Creare  \\
#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
void main (void)
{
  int  Lun;
  clrscr();
  Lun=open("Fis.Txt",O_CREAT|O_TEXT);
  if  (Lun!=-1)
     {
       cout << "Open ..." << endl;
       write(Lun,"Primul...\n",10);
       write(Lun,"Al Doilea\n",10);
     }
       else cout << " Open Incorect ! ";
  close(Lun);
  getchar();
}
```

```
                                        //  Citire  \\

#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
void main (void)
{
  int  Lun;
  char Rand[10];
  textmode(1); textbackground(BLUE); textcolor(WHITE); clrscr();
  Lun=open("Fis.Txt",O_TEXT);
  if  (Lun!=-1) {    read(Lun,Rand,10);  cout << Rand;
                     read(Lun,Rand,10);  cout << Rand;   }
       else cout << " Open Incorect ! ";
  close(Lun);
            getchar();
}
```

b) Nivelul superior

Deschiderea unui fisier (*FOpen*) :

*FILE * **fopen** (const char * cale, const char * mod) ;*

unde:

mod - poate fi "**r**" pentru citire, "**w**" - scriere, "**a**" - adaugare,
"**r+**" - modificare (citire/scriere), "**rb**" - citire binara,
"**wb**" - scriere binara, sau "**r+b**" - citire/scriere binara .

```
...
    FILE *   Pf;
    Pf = fopen ("Fisier.Dat","w");
...
```

Prelucrarea pe caractere:

```
int   putc  ( int  c,  FILE * Pf ) ;
```

```
int   getc  (FILE * Pf ) ;
```

```
int   fclose  ( FILE * Pf ) ;
```

```
#include <stdio.h>
void main (void)
{    int c;
     while ((c=getc(stdin))!=EOF)
                    putc(c,stdout);
}
```

Citirea / scrierea cu format:

```
int   fscanf  ( FILE * Pf , control, lista_var. ) ;
```

```
int   fprintf  ( FILE * Pf , control, lista_expr.) ;
```

Intrari / iesiri de siruri de caractere :

```
char  * fgets  ( char *s,  int n,  FILE * Pf ) ;
```

```
int      fputs  ( char *s, FILE * Pf ) ;
```

// Scriere cu *Stream*-uri \\

```
    #include <fstream.h>
...
void main (void)
{
 ... câmp1, câmp2, ...;
 ofstream f("Fişier.Txt");
 while (cin >> câmp1 >> câmp2 ...)
        f << câmp1 <<' '<< câmp2 << ... << endl;
 f.close();
...
}
```

// Citire cu *Stream*-uri \\

```
#include <fstream.h>
...
void main (void)
{
 ... câmp1, câmp2, ...;
 ifstream f("Fişier.Txt");                    //      sau :
 while (f >> câmp1 >> câmp2 ...)              // while ( !f.eof() )
                                              //   if ( f >> câmp1>>câmp2...)
   cout << câmp1<<' '<< câmp2 << ...
 f.close();
...
}
```

18 Mai 2013 ~ 9:00-10:30