

# 9

## UNELE ASPECTE TEHNICE REFERITOARE LA PRELUCRAREA FIȘIERELOR

Înțelegerea corectă a mecanismelor referitoare la prelucrarea fișierelor necesită cunoașterea unor detalii tehnice de realizare a operațiilor de **I/E**. Câteva din ele sunt prezentate în continuare, altele în anexa 3.

### 9.1 Realizarea fizică a transferului de date

Procedurile și funcțiile de **I/E** asigură, prin intermediul **DOS**, trecerea dintre nivelul logic, specific utilizatorului, care consideră fișierul ca o succesiune de articole (blocuri, linii, câmpuri) și nivelul fizic de organizare a datelor, în raport cu care fișierul apare ca o succesiune de octeți. În suportul extern magnetic (disc), înregistrarea datelor se face pe număr întreg de sectoare. Sub sistemul de operare MS-DOS, sectorul are 512 octeți și reprezintă unitatea de transfer cu memoria principală. Fiecărui fișier îi sunt alocate două tipuri de zone de memorie internă: zonă utilizator și zonă tampon.

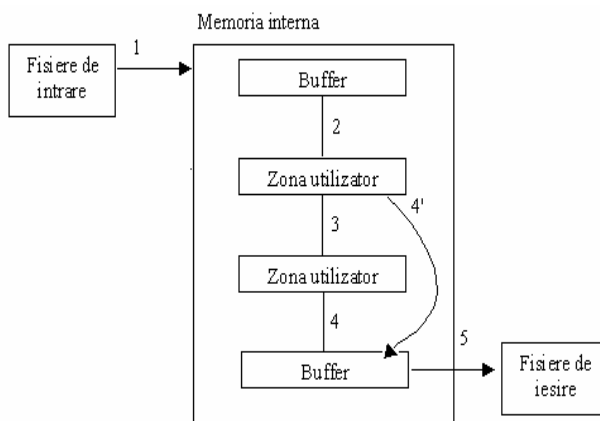
*Zona utilizator* este descrisă în **VAR** sau **CONST**. La ea are acces programul utilizatorului. Zonă are semnificații diferite, în funcție de tipul fișierului care se prelucrează. Ea este fie un articol (la fișiere cu tip), fie un bloc (la fișiere fără tip), fie este o mulțime de zone independente care sunt specificate în operațiile de citire/scriere (la fișierele **TEXT**).

*Zona tampon* (buffer) reprezintă zonă de memorie principală în/din care se face transferul datelor din/în exterior și are, sub MS-DOS, 528 octeți. Un sistem poate lucra la un moment dat cu mai multe buffer-e, numărul lor fiind stabilit la configurarea sistemului (comanda **BUFFERS** din MS-DOS). Cu cât numărul de buffere este mai mare cu atât crește viteza de transfer, dar și memoria internă ocupată. La citiri succesive, sectoarele sunt încărcate în buffere "eliberate" și de aici în zonele utilizator. Dacă, de exemplu, se citește un articol de 120 octeți, sistemul citește în buffer un sector întreg și mută, de aici, în zonă utilizator, 120 octeți. La următoarea citire, sistemul va utiliza alt

buffer etc., astfel încât, la un moment dat, buffer-ele vor conține cele mai recente date citite. Dacă citirea se face aleator (nu secvențial), se încarcă în buffer(e) sectorul/sectoarele (ca întregi) care conțin(e) articolul. Dacă articolul este deja în buffer(e), nu are loc transfer din exterior ci numai din buffer(e) în zonă utilizator.

Succesiunea de principiu a operațiilor, în cazul unui flux general de date care implică un singur buffer, este prezentată în figura 9.1. Ea este următoarea:

1. citirea unui sector din fișierul de intrare în zonă tampon asociată;
2. transferul datelor din buffer în zonă utilizator asociată fișierului de intrare;
3. pregătirea conținutului zonei utilizatorului asociată fișierului de ieșire, pe baza datelor preluate din zona fișierului de intrare sau din alte surse.



**Fig.9.1** - Fluxul general de date în operațiile de I/E

În limbajul PASCAL, aceeași zonă utilizator poate fi folosită atât pentru fișierul de intrare, cât și pentru cel de ieșire;

4. Transferul datelor din zona utilizator în buffer-ul fișierului de ieșire;
5. Scrierea în fișierul de ieșire a sectorului (când este completat), din zona tampon.

Cu toate că procesul de trecere dintre nivelurile fizic și logic are trăsături principale comune, există deosebiri esențiale de realizare pentru fișierele **TEXT** și cele binare. Pentru fișierele binare (cu tip și fără tip) se poate considera valabilă schema de principiu din figura 9.1. Transferul intern dintre buffer și zonă utilizator (operațiile 2 și 4) are loc fără conversii, iar operațiile de intrare/ieșire dintr-un program pot avea loc pe același fișier. Pentru fișierele **TEXT**, o prima particularitate constă în aceea că datele sunt transferate în/din una sau mai multe zone de memorie independente și neomogene ca tip (figura 9.2).

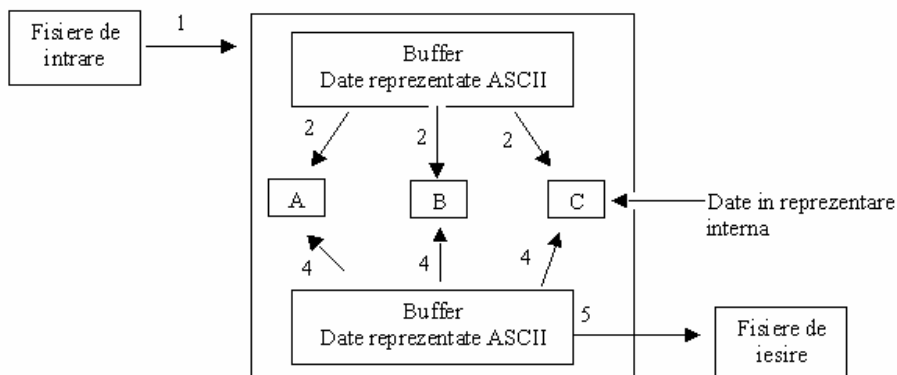


Fig.9.2 - Principiul transferului datelor în cazul fișierelor TEXT

În plus, datele sunt "decupate" din zona buffer în zonele de date, pe baza unor caractere cu rol de separator sau după alte reguli. Pentru datele numerice (întregi și reale), transferul din/în zonele buffer în/din zonele de date (operațiile 2 și 4 din figura 9.2) are loc cu conversie. Același lucru se întâmplă la scriere și cu datele de tip BOOLEAN.

Atât pentru fișierele binare cât și pentru cele **TEXT**, operațiile de transfer din exterior în buffer (1) și din acesta în zona utilizator (2) au loc, de regulă, ca urmare a execuției procedurilor de citire. În unele situații, operația (1) are loc ca urmare a execuției funcțiilor de testare a sfârșitului de fișier. Operațiile de transfer din zona utilizator în buffer (4) și din acesta în exterior (5) au loc ca urmare a execuției procedurilor de scriere.

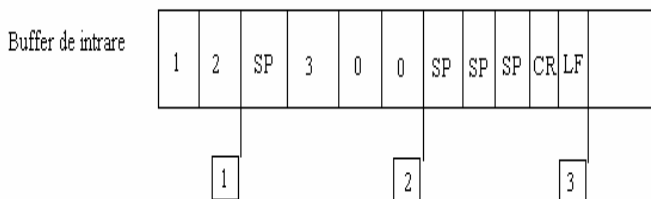
Operațiile 1 și 2 (respectiv 4 și 5) nu au loc, întotdeauna, simultan. Operația 1 are loc numai când buffer-ul de intrare este "eliberat", iar operația 5 are loc numai când buffer-ul de ieșire este plin. Procesul se desfășoară diferit la fișierele binare și la cele **TEXT**.

♦ **La fișiere binare** buffer-ul de intrare este "eliberat" ca urmare a execuției unei operații de deschidere sau a citirii unui articol/bloc (**Read, BlockRead**) din fișierul respectiv.

### Exemplu:

```

9.1. RESET(fis);  ➤ buffer "eliberat"
.....
Read(fis,art);   ➤ realizează operațiile 1 și 2 din
                  figura 9.1 (buffer "eliberat")
  
```



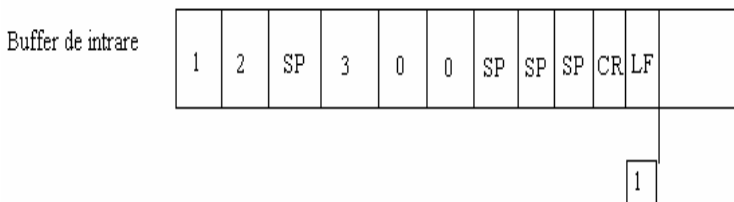
Tot procedura Read(a) realizează transferul (cu conversie) din buffer în zona **a** (a=12). Pointerul se plasează în poziția 1. Procedura Read(b) găsește bufferul "neeliberat", deci nu execută transfer din exterior, ci numai din buffer în zona **b** (b=300). Pointerul se plasează în poziția 2. Procedura Read(c) găsește buffer-ul "neeliberat", deci nu execută transfer din exterior, ci analizează caracterele din buffer. Cum la citirea datelor numerice spațiile și caracterele CR/LF sunt ignorate, pointerul avansează până în poziția 3, când buffer-ul devine "eliberat", producându-se în acest moment o cerere de transfer din exterior în buffer etc. În concluzie, secvența anterioară realizează:

```
Read(a)      ➤ operațiile 1 •i 2;  
Read(b)      ➤ operația 2;  
Read(c)      ➤ analiza, operația 1 etc..
```

9. 4. Se presupun trei variabile **a**, **b**, **c** de tip numeric și secvența de program:

```
ReadLn(a) ; {prima citire din program}  
ReadLn(b) ;  
ReadLn(c) ;
```

Înainte primului ReadLn buffer-ul de intrare este "eliberat" (datorită deschiderii implicite). De aceea, ReadLn(a) transferă date din exterior în buffer. Se presupune că au fost introduse următoarele valori:



Tot procedura ReadLn(a) transferă (cu conversie) din buffer în zona **a** (a=12) și plasează pointerul la sfârșitul liniei (în poziția 1), după CR/LF.

Procedura ReadLn(b) găsește buffer-ul "eliberat" și provoacă cerere de transfer din exterior etc.

Funcția **Eof** poate găsi buffer-ul "eliberat" sau "neeliberat". Când îl găsește "eliberat" produce un transfer din exterior în buffer, plasează pointerul pe începutul său și testează dacă este sfârșit de fișier (**CTRL/Z**). Dacă funcția **Eof** găsește buffer-ul "neeliberat", testează dacă pointerul indică sfârșitul de fișier. După testarea sfârșitului de fișier, funcția **Eof** nu modifică pointerul.

Eliberarea buffer-ului unui fișier TEXT, cu scrierea conținutului său în suportul extern, se poate realiza cu procedura **Flush(f)**. **F** trebuie deschis pentru scriere.

În concluzie, atât pentru fișierele binare, cât și pentru cele **TEXT**, trebuie reținute următoarele observații importante: • nu întotdeauna operațiile **Read**, **ReadLn**,

**BlockRead** produc transfer din exterior în memoria principală. Transferul are loc numai dacă buffer-ul este "eliberat"; • funcția **Eof** realizează și operația de transfer din exterior în memoria principală. Acest lucru se întâmplă când, la execuția funcției, buffer-ul de intrare este "eliberat".

## 9.2 Accesul la blocul de informații despre fișier

Declararea variabilei asociate fișierului, de orice tip, are ca efect rezervarea și inițializarea parțială de către compilator a unui bloc de informații (File **Informațion Block**). Unele câmpuri ale blocului primesc valori la execuția procedurii **Assign**, iar altele la deschiderea fișierului. Pentru fiecare fișier din program, compilatorul generează câte un **FIB**. Numele simbolic al blocului coincide cu numele intern asociat fișierului prin descrierea din secțiunea **VAR**. Informațiile din **FIB** sunt utilizate, la execuție, de procedurile și funcțiile care realizează operații de **I/E**.

Structura **FIB** este definită în unit-ul **Dos** prin două declarații de tip: **FileRec**, pentru fișierele binare și **TextRec**, pentru fișierele **TEXT**.

♦ **Tipul FileRec** reprezintă structura **FIB** pentru fișiere cu tip și fără tip. El este definit astfel:

```
FileRec = RECORD
    Handle   : WORD;
    Mode     : WORD;
    RecSize  : WORD;
    Private  : ARRAY[1...16] OF BYTE;
    UserData : ARRAY[1...16] OF BYTE;
    Name     : ARRAY[0...79] OF CHAR;
END;
```

♦ **Tipul TextRec** reprezintă structura **FIB** pentru fișierele **TEXT**. El este definit astfel:

```
TextRec = RECORD
    Handle      : WORD;
    Mode        : WORD;
    BufSize     : WORD;
    Private     : WORD;
    BufPos      : WORD;
    BufEnd      : WORD;
    BufPtr      : ^TextBuf;
```

OpenFunc	: Pointer;
InOutFunc	: Pointer;
FlushFunc	: Pointer;
UserData	: ARRAY[1...16] OF BYTE;
Name	: ARRAY[0...79] OF CHAR;
Buffer	: TextBuf;

Semnificaţia principalelor câmpuri este următoarea:

- **Handle** este o valoare întregă care reprezintă identificatorul pentru fişiere deschise. Valorile 1-7 sunt utilizate pentru dispozitivele standard de **I/E** (intrare, ieşire, auxiliar, imprimantă, fişierele în curs de folosire cu comanda **PRINT** din **DOS** şi de reţea). Fişierele deschise ale utilizatorului au valori pentru **handle** începând cu 8. Fişierele nedeschise au **handle=0**. O valoare **handle** asociată unui fişier devine disponibilă la închiderea acestuia.

### **Exemple:**

9.5. Dacă într-un program se lucrează simultan cu trei fişiere, valorile **handle** asociate sunt 8, 9, 10.

9.6. Dacă într-un program se lucrează cu trei fişiere deschise şi închise pe rând, fiecare va avea valoarea **handle** opt.

Într-un program pot fi deschise simultan maxim 12 fişiere ale utilizatorului (**handle** cu valori din intervalul [8, 19]).

- **Mode** indică starea fişierului, care poate fi exprimată şi prin următoarele constante definite în unit-ul **Dos**:

FmClosed= \$D7B0	➤ fişier închis;
FmInput = \$D7B1	➤ fişier deschis pentru citire;
FmOutput= \$D7B2	➤ fişier deschis pentru scriere;
FmInOut = \$D7B3	➤ fişier deschis pentru citire/scriere;

Fişierele **TEXT** pot avea stările FmClosed, FmInput, FmOutput. Fişierele binare pot avea orice stare (implicit FmClosed sau FmInOut). În unit-ul **System** este definită variabila **FileMode** astfel:

FileMode:BYTE=2

Variabila determină modul de deschidere a fişierelor binare de către procedura **Reset**. Valorile ei posibile sunt: **0** - fişier FmInput; **1** - fişier FmOutput; **2** - fişier FmInOut. Variabila are valoarea implicită 2. Atribuirea altei valori are ca efect folosirea acestui mod de către toate apelurile ulterioare ale procedurii **Reset**.

- **RecSize** indică lungimea articolului (blocului) rezultată din descrierea internă programului.

- **Name** este identificatorul extern al fișierului, așa cum este precizat de procedura **Assign**.

- **BufSize** reprezintă lungimea buffer-ului fișierului **TEXT**.

- **BufPos**, **BufEnd**, **BufPtr** reprezintă pointeri folosiți în gestiunea buffer-elor asociate fișierului **TEXT**.

- **OpenFunc**, **InOutFunc**, **CloseFunc**, **FlushFunc** reprezintă adresa **driver**-elor pentru deschidere, citire/scriere, închidere, golire a buffer-ului.

- **Buffer** reprezintă zonă tampon asociată fișierului (chiar buffer-ul fișierului).

Tipul **TextBuf** este definit în unit-ul **Dos** astfel: **TextBuf** = **ARRAY**[0...127] of **CHAR**.

Pentru a avea acces la informațiile din **FIB**, trebuie declarată o variabilă de tip **FileRec** (respectiv **TextRec**) care să aibă aceeași adresă cu **FIB**-ul (cu variabila de tip fișier):

a) Pentru fișiere **TEXT**:

VAR

f: **TEXT**; inf\_f: **TextRec** **ABSOLUTE** f;

b) Pentru fișiere cu tip:

VAR

f: **FILE OF tip**; inf\_f: **FileRec** **ABSOLUTE** f;

c) Pentru fișiere fără tip:

VAR

f: **FILE**; inf\_f: **FileRec** **ABSOLUTE** f;

Câmpurile zonei **inf\_f** se adresează cu denumirile lor din definirea articolului **FileRec**, respectiv **TextRec**, din unit-ul **Dos**.

### 9.3 Variabile și funcții pentru prelucrarea erorilor de I/E

În fiecare dintre unit-urile standard sunt definite variabile și funcții care pot semnaliza modul de desfășurare a operațiilor de intrare/ieșire.

- ◆ În unit-ul **System** sunt definite următoarele variabile și funcții:

- Variabila **InOutRes** conține codurile de eroare la execuție, generate de rutinele de **I/E**, corespunzând unor situații cum sunt (anexa 3): eroare la citire de pe disc (codul 100), la scriere pe disc (101), fișier neassignat (102), fișier nedeschis (103), fișier nedeschis pentru intrare (104), fișier nedeschis pentru ieșire (105), format numeric invalid (106). Variabila **InOutRes** are valoarea zero dacă operația de **I/E** s-a desfășurat normal. Ea este definită astfel: **InOutRes**:**Integer**=0;

- Funcția **IOResult** returnează programului valoarea variabilei **InOutRes** și o



pune pe aceasta pe zero. Dacă valoarea **InOutRes** este diferită de zero și directiva de compilare **\$I** are valoarea **{\$I-}**, programul nu se întrerupe, dar următoarea operație de **I/E** nu se va mai executa; dacă are valoarea **{\$I+}** programul se oprește cu eroare de execuție. De aceea, dacă se dorește controlul desfășurării unei operații de **I/E**, se procedează astfel (pe exemplul procedurii **READ**):

```
{ $I- }      { inhibă întreruperea programului }
Read (f,zonă);
{ $I+ }      { autorizează execuția următoarei operații de I/E }
IF IOResult <> 0
THEN        { eroare de I/E } ELSE { nu există eroare de I/E };
```

- Variabila **ErrorAddr** conține adresa unde s-a produs eroarea de execuție. În cazul inexistenței erorii, conține valoarea **nil**. Variabila este definită astfel: **ErrorAddr:pointer=nil**. Adresa este memorată sub forma **ssss:dddd**, unde **ssss** este adresa segmentului de cod, iar **dddd** este offset-ul (deplasarea în cadrul segmentului).

- ♦ În unit-ul **Dos** sunt definite următoarele proceduri și variabile:

- Variabila **DosError**, de tip **INTEGER**, este inițializată de funcțiile și procedurile din acest unit, cu următoarele valori principale (vezi anexa 1):

- 0 - fără eroare;
- 2 - fișier negăsit;
- 3 - cale negăsită;
- 4 - prea multe fișiere deschise;
- 5 - acces interzis; etc.

Variabila **DosError** are valoarea zero când execuția funcțiilor și a procedurilor definite în unit-ul **Dos** se termină normal.

- Procedura **SetVerify** poziționează comutatorul **verify** din **DOS**. Comutatorul are două valori posibile: **ON** (valoare logică **TRUE**) sau **OFF** (valoare logică **FALSE**). Când comutatorul are valoarea **ON**, sistemul de operare face verificare după fiecare operație de scriere în fișiere (se verifică dacă datele scrise pot fi citite fără eroare). Poziția **On** a comutatorului **verify** mărește timpul de execuție a programelor. Când comutatorul are valoarea **OFF**, sistemul de operare nu face verificarea scrierii. Valoarea implicită a comutatorului este **OFF**. Valoarea comutatorului rămâne activă până la o nouă setare. Procedura este defintă astfel:

#### ***SetVerify(v:Boolean)***

**V** este o variabilă de tip **BOOLEAN**. Când **v** are valoarea **TRUE**, comutatorul **verify** primește valoarea **ON**. În caz contrar primește valoarea **OFF**. Procedura are efect similar cu comanda **VERIFY** din **DOS**. Dacă comutatorul **verify** este **ON**, rezultatul verificării scrierii este memorat în variabila **DosError**.

În anexa 3 sunt prezentate principalele erori de execuție, care includ și pe cele de intrare/ieșire.

## 9.4 Anomalii în tratarea lungimii articolelor (blocurilor) fișierelor binare

Deoarece peste un fișier fizic poate fi suprapus orice tip de fișier, rămâne în sarcina programatorului să asigure compatibilitatea cu cerințele de prelucrare. În cazul fișierelor binare, lungimea și structura articolelor (blocurilor) pot diferi între momentul creării și cel al exploatării.

### Exemplu:

9.7. La momentul creării, o matrice este scrisă câte o linie pe articol:

```
TYPE
  a=ARRAY[1..10] of REAL;
VAR
  Fis:FILE OF a;
  Linie:a;
  .....
BEGIN
  Assign(Fis, 'MATRICE.DAT');
  Rewrite(Fis);
  .....
  Write(Fis, Linie);
  .....
```

La momentul exploatării, matricea poate fi citită câte un element pe articol:

```
VAR
  Fis:FILE OF REAL;
  Element:REAL;
  .....
BEGIN
  Assign(Fis, 'MATRICE.DAT');
  Reset(Fis);
  .....
  Read(Fis, Element);
  .....
```

Fie **lart** și **lbloc** lungimea articolelor, respectiv blocurilor unui fișier binar care are lungimea **lfis**, exprimată în octeți. O prelucrare corectă a fișierului presupune să fie îndeplinite următoarele condiții: **lfis MOD lart=0**, respectiv **lfis MOD lbloc=0**.

În caz contrar se produc anomalii în prelucrare, ultimii **lfis MOD lart**, respectiv **lfis MOD lbloc** octeți, neputând fi prelucrați.

### Exemplu:

9.8. În programul demonstrativ care urmează se creează fișierul TEST.DAT care are lungimea 8 octeți (se scriu patru articole de câte doi octeți). Fișierul este exploatat cu

articole de 6 octeți (REAL), **FileSize(f2)** returnând valoarea 1 (8 DIV 6). Prima citire din fișier transferă primii 6 octeți. A doua citire produce eroare de I/E (8 MOD 6 = 2).

```
PROGRAM Ex8A;
VAR
  f1:FILE OF WORD;
  x:WORD;
  f2:FILE OF REAL;
  y:REAL;
BEGIN
  Assign(f1,'TEST.DAT'); Rewrite(f1);
  x:=0;
  Write(f1,x,x,x,x);
  Close(f1);
  Assign(f2,'TEST.DAT'); Reset(f2);
  Writeln(FileSize(f2));
  Read(f2,y);
  Read(f2,y); {Error 100: Disk read error}
  Close(f2);
END.
```

Este interesant de analizat și modul în care lucrează funcția **Eof(f)**, în situația fișierului anterior. Dacă, după prima citire (6 octeți), se apelează funcția **Eof(f2)**, aceasta returnează valoarea FALSE (programul de mai jos afișează textul FALSE), cu toate că pointerul este pe articolul cu numărul relativ **FileSize(f2)** ( $\text{FileSize}(f2) = \text{lfs} \text{ DIV lart} = 1$ ).

```
PROGRAM EX8B;
VAR
  f1:FILE OF WORD;
  x:WORD;
  f2:FILE OF REAL;
  y:REAL;
BEGIN
  Assign(f1,'TEST.DAT'); Rewrite(f1);
  x:=0;
  Write(f1,x,x,x,x);
  Close(f1);
  Assign(f2,'TEST.DAT'); Reset(f2);
  Read(f2,y);
  IF Eof(f2) THEN BEGIN Writeln('TRUE'); Readln END
  ELSE BEGIN Writeln('FALSE'); Readln END;
  Close(f2);
END.
```

Din exemplul anterior se desprinde concluzia că funcția **Eof(f)** returnează valoarea TRUE dacă pointerul este plasat după **lfs** octeți față de începutul fișierului. Când nu se produce anomalie de lungime, afirmația anterioară coincide cu faptul că **Eof(f)** returnează valoarea TRUE când pointerul este pe articolul cu numărul relativ **FileSize(f)** ( $\text{FileSize}(f) \cdot \text{lart} = \text{lfs}$ ). De fapt, în cazul anomaliei de lungime, funcția

**Eof(f)** nu poate returna niciodată valoarea TRUE, deoarece "articolul scurt" nu poate fi niciodată citit. Din cele prezentate anterior, rezultă că, în cazul prelucrării unui fișier cu structură necunoscută, este recomandabil să se lucreze cu articole de tip CHAR sau cu blocuri de lungime unu.