Şiruri de caractere[*)]

Capitolul **10**

- **Operaţii cu variabile de tip `string`**
- **Subprograme predefinite pentru *string*-uri**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluţiile problemelor**

Tipul *şir de caractere* se identifică în Pascal prin termenul *string*, deoarece în acest limbaj de programare există tipul predefinit **`string`** pentru tablouri unidimensionale care au elemente câte un caracter. Diferenţele importante dintre un şir de caractere şi un *string* se desprind din următoarele observaţii:

- Unui tablou i se specifică de la început numărul maxim de componente pe care poate să le conţină, pe când o variabilă de tip **`string`** are implicit cel mult 255 de elemente.
- Dacă la declarare se specifică pentru dimensiunea maximă a unei variabile de tip **`string`** un număr mai mic decât 255, va fi valabilă declaraţia făcută explicit.
- Un *string* se memorează pe atâţia octeţi câte caractere sunt specificate în declarare, la care se adaugă un octet în care se memorează lungimea *string*-ului. Acest lucru nu se întâmplă la tablourile de caractere care se memorează pe exact atâţia octeţi câte caractere apar în declarare.

***Declararea variabilelor de tip* `string`**

Cu toate că tipul descrie o structură (şir de caractere), nu este obligatoriu să precizăm o anumită lungime, dar dacă dorim să declarăm un *string* mai scurt decât lungimea implicită (255), atunci acest lucru este posibil.

Exemplu

```
var fraza:string;          { variabila fraza are cel mult 255 de caractere }
    cuv:string[20];        { variabila cuv poate avea cel mult 20 de caractere }
```

[*)] În cele ce urmează, atunci când vom scrie *string*, în acest material, ne referim la un şir de caractere. Atunci când vom scrie **`string`**, este vorba de tipul predefinit în limbajul Pascal.

10.1. Operații cu variabile de tip `string`

10.1.1. Citirea

O variabilă de tip **`string`** se poate citi în Pascal, specificându-i numele. Avantajul față de tablouri constă în faptul că nu trebuie să ştim exact numărul caracterelor. În momentul în care am apăsat **Enter**, sau, în caz de citire dintr-un fişier de tip `Text`, atunci când urmează caracterul sfârşit de linie, *string*-ul „s-a terminat" şi prima componentă a variabilei (cea cu indice 0) primeşte o valoare egală cu numărul caracterelor citite.

Exemplu

```
var s:string;
    ss:string[5];
...
  ReadLn(s);
  ReadLn(ss);
...
```

Dacă pentru *s* şi pentru *ss* introducem şirul de caractere `'ABC'`, valoarea ambelor variabile va fi `'ABC'`. Dacă pentru *ss* introducem `'ABCABC'`, valoarea lui *ss* va fi `'ABCAB'`, adică se vor păstra cel mult atâtea elemente câte permite declarația.

10.1.2. Afişarea

Afişarea se realizează specificând o expresie de tip **`string`**.

Exemplu

```
var s:string;
    ss:string[5];
...
  WriteLn('ABCABC');
  WriteLn(s+ss);      { dacă s = 'str' şi ss = 'ing', se va afişa 'string' }
...
```

10.1.3. Atribuirea

Unei variabile de tip **`string`** i se poate atribui valoarea unei expresii care are tipul **`string`**. Dacă variabila are un tip care precizează o lungime mai mică decât *string*-ul din expresie, caracterele care depăşesc lungimea variabilei se pierd.

Exemplu

Fie declarația:

```
var ss:string[5];
```

Dacă valoarea lui `ss='123'`, atribuirea `ss:=ss+'457890';` se va finaliza cu valoarea `'12345'` pentru `ss`.

10.1.4. Concatenarea

Concatenarea se poate realiza utilizând operatorul `'+'`, dar în Pascal există şi o procedură predefinită în acest scop (`Concat`).

Exemplu

```
s3:=s1 + s2;  { dacă s1 = 'a' şi s2 = 'b', atunci valoarea lui s3 va fi 'ab' }
s3:=s3 + '***' + s1;                    { valoarea lui s3 va fi 'ab***a' }
```

Să observăm că un *string* poate fi „construit” prin adăugare de caractere la un *string* iniţial vid.

Exemplu

```
var s:string[5];
    i:Byte;
Begin
  s:='';                                                  { stringul vid }
  i:=0;
  while i<5 do begin
    Inc(i);
    s:=s + 'a'
  end;
  WriteLn(Ord(s[0]));  { componenta care păstrează lungimea string-ului (5) }
  WriteLn(s)                                          { se va afişa 'aaaaa'}
End.
```

10.1.5. Operaţii relaţionale

În cazul expresiilor de tip **`string`** se pot folosi operatorii relaţionali cunoscuţi:
<, <=, <>, >, >=, =.

Rezultatul a două comparaţii este o constantă logică (*adevărat* sau *fals*).

Exemplu

```
var s1,s2,s3:string;
```

Având declaraţiile de mai sus, în tabelul următor prezentăm câteva exemple de expresii relaţionale după efectuarea următoarelor operaţii de atribuire:

```
s1:='A';
s2:='ABC';
s3:=s1 + s2;
```

Expresie	Rezultat
`s1 = 'A'`	`true` (s-a comparat `'A'` cu `'A'`)
`s2 <> 'ABC'`	`false` (s-a comparat `'ABC'` cu `'ABC'`)
`s3 < 'ABC'`	`true` (s-a comparat `'AABC'` cu `'ABC'` şi `'A' < 'B'` pe cea de a doua poziţie)
`s1 < 'AA'`	`true` (s-a comparat `'A'` cu `'AA'`)
`s3 >= s1`	`true` (s-a comparat `'AABC'` cu `'A'`)

10.1.6. Accesul la o componentă

Componenta `x[i]` reprezintă al `i`-lea caracter din variabila `x`, de tip **`string`**.

10.2. Subprograme predefinite pentru *string*-uri

Unit-ul *System* din Borland Pascal 7.0 conţine mai multe subprograme predefinite care pot fi utilizate în prelucrarea *string*-urilor. Fie *s* şi *ss* variabile de tip **`string`**.

- Funcţia `Pos(ss,s)` returnează valoarea poziţiei începând de la care se regăseşte *ss* în *s*; dacă *ss* nu se găseşte în *s*, atunci *i* ia valoarea 0.
- Funcţia `Copy(s,i,j)` returnează *substring*-ul din *s* care începe la poziţia *i* şi are *j* caractere. Dacă în *s* nu există *j* caractere începând cu poziţia *i*, se vor returna atâtea câte sunt.
- Funcţia `Length(s)` returnează lungimea *stringului* (numărul caracterelor din *s*).
- Procedura `Delete(s,i,j)` are ca efect ştergerea din *s*, începând de la poziţia *i*, a *j* caractere. Dacă în *s* nu există *j* caractere începând cu poziţia *i*, se vor şterge atâtea câte sunt.
- Procedura `Insert(ss,s,i)` inserează caracterele din *string*-ul *ss* în *s*, începând de la poziţia *i*.

10.3. Implementări sugerate

Pentru a vă familiariza cu prelucrările şirurilor de caractere, implementaţi programele aferente următoarelor exerciţii:

1. citirea unui număr caracter cu caracter;
2. citirea unui cuvânt caracter cu caracter;
3. numărarea literelor dintr-un text dat;
4. numărarea cuvintelor dintr-un text dat;
5. numărarea propoziţiilor dintr-un text dat;
6. descompunerea în cuvinte a unui text dat;
7. ordonarea unui şir de cuvinte;
8. eliminarea spaţiilor duble dintre cuvintele unui text dat.

10.4. Probleme propuse

10.4.1. Litere

Fie un text citit dintr-un fişier. Să se afişeze acele cuvinte din text în componența cărora numărul majusculelor este strict mai mare decât numărul literelor mici.

Date de intrare
Fişierul de intrare **`LITERE.IN`** conține textul scris pe mai multe linii.

Date de ieşire
Cuvintele se vor scrie în fişierul de ieşire **`LITERE.OUT`** pe linii distincte, fiecare cuvânt fiind scris pe câte o linie nouă.

Restricții şi precizări
- cuvintele sunt separate prin spații şi/sau caractere de sfârşit de linie;
- fişierul nu conține linii vide;
- liniile conțin cuvinte nedespărțite;
- pe o linie sunt scrise cel mult 255 de caractere;
- un cuvânt poate avea cel mult 255 de caractere;
- cuvintele conțin numai litere mari şi mici.

Exemplu

`LITERE.IN`		**`LITERE.OUT`**
`Ana AVEA MULte`	`Mere`	`AVEA`
`  ACESTa`		`MULte`
`esTe un eXeMPLu`		`ACESTa`
		`eXeMPLu`

Explicații
- `Ana` are două litere mici şi una mare, deci nu se scrie.
- `AVEA` are patru litere mari şi nici una mică, deci se scrie.
- `MULte` are trei litere mari şi două mici, deci se scrie.
- `Mere` are o literă mare şi trei mici, deci nu se scrie.
- `ACESTa` are patru litere mari şi o literă mică, deci se scrie.
- `esTe` are o literă mare şi trei litere mici, deci nu se scrie.
- `un` nu are nici o literă mare, deci nu se scrie.
- `eXeMPLu` are 4 litere mari şi trei litere mici, deci se scrie.

10.4.2. Text

Se citesc mai multe propoziţii dintr-un fişier. Se cere:

- să se afişeze numărul de cuvinte din fiecare propoziţie;
- să se determine pentru fiecare propoziţie numărul cuvintelor distincte;
- să se afişeze cuvintele care se repetă în cadrul aceleiaşi propoziţii.

Date de intrare

Propoziţiile sunt scrise în fişierul de intrare **`TEXT.IN`** pe mai multe linii. Două propoziţii sunt separate prin caracterele `'.'`, `'!'`, `'?'`.

Date de ieşire

Rezultatele se vor scrie în fişierul de ieşire **`TEXT.OUT`**. Pe linia *i* din fişier se afişează datele referitoare la a *i*-a propoziţie. Aceste date se vor separa prin spaţiu şi vor apărea în ordinea: *numărul cuvintelor* din propoziţie, *numărul cuvintelor distincte*, *lista cuvintelor* care nu sunt distincte în cadrul propoziţiei.

Restricţii şi precizări

- din cuvinte poate să facă parte caracterul `'-'`;
- două cuvinte se separă prin caracterele: `' '`, `':'`, `','`;
- pot exista separatori inutili;
- numărul maxim de cuvinte pentru fiecare propoziţie este 30;
- în text există litere mari şi mici.

Exemplu

`TEXT.IN`

```
Sa revenim la coroana pe care am lasat-o in mana lui Arhimede,
care se uita ganditor la ea nestiind ce sa-i faca.
O lasa deceptionat din mana si pleaca la alte treburi.
Dar problema, ca intrebare, staruie in mintea lui; s-o
cantaresc, s-o cantaresc, aude in sine un murmur in surdina,
da, asta stiu, s-o cantaresc, dar ce altceva sa-i mai fac?
```

`TEXT.OUT`

```
22 18 la care
10 10
30 21 in s-o cantaresc
```

10.4.3. Traducere

Un elev are de scris un text în limba engleză, dar nu cunoaşte destul de bine această limbă. Se hotărăşte să lase cuvintele pe care nu le poate traduce în limba română, apoi

să le caute în dicţionar. Din aceste cuvinte va alcătui o listă, urmând ca ulterior să le înlocuiască în text.

Date de intrare
Pe prima linie a fişierului **`TRADUS.IN`** se află textul elevului, iar pe fiecare dintre următoarele linii se află un cuvânt în limba română, urmat, după un spaţiu, de corespondentul său în limba engleză.

Date de ieşire
Textul tradus, împreună cu semnele de punctuaţie existente în textul dat în fişierul de intrare, se va afişa în fişierul de ieşire **`TRADUS.OUT`**.

Restricţii şi precizări
- textul în fişierul de intrare trebuie să conţină cel mult 255 de caractere;
- separatorii folosiţi în text pot fi `' '`, `'.'`, `','`;
- textul din fişierul de ieşire va avea exact aceeaşi separatori ca şi cel de intrare.

Exemplu
`TRADUS.IN`
```
Ana has many mere, dar she is going to the piata to vanda them.
mere apples
dar but
piata marketplace
vanda sell
```

`TRADUS.OUT`
```
Ana has many apples, but she is going to the marketplace to sell them.
```

10.4.4. Culori
La o grădiniţă se va realiza un spectacol, unde copiii trebuie să fie îmbrăcaţi în costume de culori diferite. Fiecare copil îşi spune culoarea preferată, iar învăţătoarea formează grupuri de copii îmbrăcaţi în costume de aceeaşi culoare. Pe scenă copiii se vor aşeza în grupuri, astfel încât cel mai numeros grup va sta în fundal, în faţa acestora va sta grupul din care fac parte mai puţini copii etc., iar în faţă grupul cel mai puţin numeros.

Date de intrare
Datele de intrare se citesc din fişierul **`CULORI.IN`** care are următoarea structură:
- pe prima linie se află numărul natural *n*, reprezentând numărul de copii;
- pe următoarele *n* linii sunt scrise numele şi prenumele copiilor; numele este separat de prenume printr-un spaţiu;

- pe următoarea linie este scris numărul natural *m*, reprezentând numărul de culori;
- pe următoarele *n* linii sunt trecute culorile preferate ale celor *n* copii sub forma unor numere naturale din intervalul [1, *m*] cu care s-au codificat culorile; fiecare copil alege o singură culoare.

Date de ieşire

Datele de ieşire se scriu în fişierul **CULORI.OUT** pe *m* linii. Fiecare linie conţine codul culorii, urmat de numele copiilor care preferă culoarea respectivă. Numele vor fi despărţite printr-un spaţiu. Numărul numelor va fi cel mai mare pe prima linie, mai mic pe linia a doua şi cel mai mic pe ultima linie.

Restricţii şi precizări

- $2 \le n \le 20$;
- $2 \le m \le 7$;
- fiecare culoare dintre cele *m* va fi preferată de cel puţin un copil;
- în cazul în care există două sau mai multe culori preferate de un număr egal de copii, atunci ordinea în care grupurile corespunzătoare se vor aşeza pe scenă nu contează.

Exemplu

CULORI.IN
```
7
Pop Mihai
Popan Ana
Popa Ion
Popescu Marius
Rus Andrei
Rusu Teodor
Trif Ionut
3
1
1
2
3
3
3
2
```

CULORI.OUT
```
3 Popescu Marius Rus Andrei Rusu Teodor
1 Pop Mihai Popan Ana
2 Popa Ion Trif Ionut
```

10.5. Soluțiile problemelor propuse

10.5.1. Litere

Vom citi câte un *string* de pe o linie din fişier şi îl vom parcurge în felul următor: vom ignora spațiile până la apariția primului caracter diferit de spațiu. Apoi, vom construi din caractere consecutive litere un cuvânt, numărând în paralel şi majusculele. În momentul în care apare primul spațiu, sau am ajuns la sfârşitul *string*-ului, prelucrăm cuvântul, respectiv decidem dacă trebuie să-l scriem în fişier sau nu. Reluăm algoritmul pentru restul caracterelor din *string*-ul citit, apoi pentru restul liniilor din fişier.

```
Subalgoritm Litere:
  cât timp nu urmează marca de sfârşit de fişier execută:
    citeşte t                                          { citim o linie din fişier }
    i ← 1                                          { vom parcurge stringul citit }
    cât timp i ≤ lungimea textului t execută:
      cât timp (i ≤ lungimea textului t) şi (t[i]=' ') execută:
        i ← i + 1                                        { spațiile le ignorăm }
      cuv ← ''                                  { vom construi cuvântul curent }
      maj ← 0
      cât timp (i ≤ lungimea textului t) şi (t[i] ≠ ' ') execută:
        dacă (t[i] ≥ 'A') şi (t[i] ≤ 'Z') atunci          { majusculă }
          maj ← maj + 1
        sfârşit dacă
        cuv ← cuv + t[i] { concatenăm caracterul curent în cuvântul curent }
        i ← i + 1
      sfârşit cât timp
      dacă maj > [(lungimea cuvântului cuv)/2] atunci
        scrie cuv
      sfârşit dacă
    sfârşit cât timp
  sfârşit cât timp
sfârşit subalgoritm
```

10.5.2. Text

Vom reține fiecare propoziție într-un şir de caractere şi vom construi şirul cuvintelor care fac parte din ea pentru a facilita căutarea cuvintelor care nu sunt unice în cadrul propoziției.

Din cauza că o propoziție se poate continua în fişierul de intrare pe mai multe linii, vom forma propozițiile, citind din fişier caractere. Notăm cu *pr* şirul de caractere care reține propoziția curentă şi cu *c* caracterul citit din fişier. Inițializăm propoziția cu

stringul vid şi caracterele citite le concatenăm în *pr*, până la apariţia unui separator sau a marcajului de sfârşit de linie.

- În cazul în care propoziţia se încheie (a apărut un separator) înlocuim separatorul, adăugând la sfârşitul variabilei *pr* un spaţiu, pentru a simplifica împărţirea pe cuvinte. Trecem la analiza propoziţiei.
- Dacă pe parcursul citirii a apărut marcaj de sfârşit de linie, fără ca propoziţia să se termine, acesta se citeşte din fişier, continuând citirea caracterelor şi concatenarea lor în aceeaşi propoziţie de pe linia următoare.

Algoritmul care realizează formarea unei propoziţii este:

```
Algoritm Text:
  pr←''                                    { iniţializarea primei propoziţiei }
  cât timp nu urmează marca de sfârşit de fişier execută:
    citeşte c
      dacă c nu este separator de propoziţii şi
                                  nu urmează marca de sfârşit de linie atunci
        pr ← pr + c                  { alipim caracterul curent propoziţiei }
      altfel
        dacă c este separator de propoziţii atunci
          pr ← pr + ' '              { suprascriem separatorul cu spaţiu }
          Analiza(pr)                              { analiza propoziţiei }
          scrie marca de sfârşit de linie
          pr ← ''                    { iniţializarea următoarei propoziţii }
        altfel
          citeşte marca de sfârşit de linie
       sfârşit dacă
     sfârşit dacă
  sfârşit cât timp
sfârşit algoritm
```

Analiza propoziţiei reţinute în şirul de caractere *pr* începe cu înlocuirea separatorilor cu spaţii, apoi se elimină spaţiile inutile şi se formează tabloul de cuvinte. Se scrie numărul *n* al cuvintelor depistate în fişierul de ieşire. Algoritmul `Analiza(pr)` apelează subalgoritmul `distincte(cuvinte,dist)` care numără cuvintele care apar o singură dată în propoziţie şi scrie acest număr în fişierul de ieşire.

De asemenea, se ţine şi evidenţa cuvintelor care se repetă şi începând de la a doua apariţie se înlocuiesc cu spaţiu pentru a nu le mai lua în calcul.

După scrierea celor două numere în fişierul de ieşire se trece la căutarea cuvintelor care se repetă. Se numără fiecare apariţie şi se înlocuieşte dublura lui cu spaţiu. Dacă se găsesc cuvinte care se repetă, acestea se vor scrie în fişierul de ieşire.

```
Subalgoritm Analiza(pr):                          { analizăm o propoziţie }
  suprascriem separatorii cu spaţii
  eliminăm spaţiile în plus
  căutăm cuvinte, le reţinem în şirul cuvinte şi le ştergem din propoziţie
  scrie n,' '                                  { numărul cuvintelor din propoziţie }
  distincte(cuvinte,dist)                     { apelăm subalgoritmul distincte }
  scrie dist,' '
  pentru i=1,n-1 execută:
    apare ← 1                          { numărul cuvintelor care nu sunt distincte }
    dacă cuvinte[i] ≠ ' ' atunci
      pentru j=i+1,n execută:
        dacă cuvinte[i] = cuvinte[j] atunci
                              { un cuvânt este înlocuit cu spaţiu la a doua apariţie }
          cuvinte[j] ← ' '
          apare ← apare + 1
        sfârşit dacă
      sfârşit pentru
      dacă apare > 1 atunci                        { cuvântul nu este distinct }
        scrie a[i],' '
      sfârşit dacă
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

În algoritmul care calculează numărul de cuvinte care apar o singură dată în propoziţia *pr* se numără apariţiile fiecărui cuvânt în şirul cuvintelor. Dacă acesta apare o singură dată se incrementează contorul cuvintelor distincte.

```
Subalgoritm Distincte(cuvinte,dist)
  dist ← 0
  pentru i=1,n execută:
    apariţii ← 0
    pentru j=1,n execută:
      dacă cuvinte[i] = cuvinte[j] atunci
        apariţii ← apariţii + 1
      sfârşit dacă
    sfârşit pentru
    dacă apariţii = 1 atunci  { dacă cuvinte[i] a apărut o singură dată }
      dist ← dist + 1                              { avem un cuvânt distinct }
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

10.5.3. Traducere

Dificultatea acestei probleme constă în faptul că din textul iniţial numai anumite cuvinte se înlocuiesc. Cele care trebuie înlocuite şi traducerile lor apar în fişierul de intrare începând de pe linia a doua.

Se parcurge textul iniţial *t* şi se reţin pe rând cuvintele în variabila *cuvânt*. Pentru fiecare cuvânt *cuvânt* se verifică dacă acesta apare printre cuvintele care sunt scrise în dicţionar. Dacă acesta există în dicţionar, se reţine traducerea lui, urmând ca aceasta să substituie cuvântul în textul iniţial.

Căutăm cuvântul *cuvânt* în fişierul care conţine dicţionarul cu următorul subalgoritm:

```
Subalgoritm Caută(cuvânt):
  citeşte marcajul de sfârşit de linie din fişierul de intrare
  cât timp nu urmează marca de sfârşit de fişier execută:
    citim în dictio o linie din fişier (o pereche de cuvinte)
    căutăm cuvântul cuvânt în dictio
    dacă l-am găsit începând cu poziţia 1 atunci
      ştergem din perechea de cuvinte originalul
      returnăm traducerea
      ieşim forţat din cât timp
    altfel
      returnăm stringul vid
    sfârşit dacă
  sfârşit cât timp
sfârşit subalgoritm
```

În fişierul de intrare prima linie este ocupată de textul care trebuie tradus. De fiecare dată când se caută traducerea unui cuvânt în fişier trebuie să se realizeze saltul peste prima linie, deci se citeşte doar marcajul de sfârşit de linie.

Selectarea cuvintelor care se caută în dicţionar se face în felul următor:

```
Subalgoritm Traducere(t):
  cuvânt ← ''
  i ← 1
  cât timp i ≤ lungimea textului t execută:
    cât timp t[i] este literă execută:
      cuvânt ← cuvânt + t[i]      { în cuvânt construim cuvinte din text }
      i ← i + 1
    sfârşit cât timp
```

```
    trad ← traducerea lui cuvânt, returnat de subalgoritmul Caută(cuvânt)
    dacă trad ≠ '' atunci                          { dacă există traducere }
      inserăm traducerea cuvântului în t, începând cu poziția i
      ştergem cuvântul de tradus din t
      i ← i + lungimea cuvântului trad - lungimea cuvântului cuvânt + 1
                        { recalculăm poziția din care continuăm prelucrarea lui t }
      cuvânt ← '' { reinițializăm cuvânt pentru a fi util pentru un cuvânt nou }
    altfel
      i ← i+1                              { avansăm, cuvântul nu trebuie tradus }
      cuvânt ← ''
    sfârşit dacă
  sfârşit cât timp
sfârşit subalgoritm
```

10.5.4. Culori

În rezolvare variabila *num* reprezintă tabloul care conține numele celor *n* copii, în tabloul *pref* păstrăm tabloul care conține numărul de ordine a culorii preferate de copii, iar *cul* este tabloul în care se reține numărul de apariții a fiecărei culori.

Exemplu

Fie tabloul *num* = (Pop Mihai, Popan Ana, Popa Ion, Popescu Marius, Rus Andrei, Rusu Teodor, Trif Ionut). Presupunem că există 3 culori (*m* = 3), iar şirul culorilor preferate de copii, în ordinea numelor din tabloul *num* este *pref* = (1, 1, 1, 1, 3, 3, 2).

Elementele tabloului *cul* se determină în paralel cu citirea. Acesta va conține pe poziția *i* numărul copiilor care preferă culoarea *i*, deci *cul* = (4, 1, 2) cu semnificația: culoarea 1 apare de patru ori, culoarea 2 apare o dată, culoarea 3 apare de două ori.

```
Subalgoritm Citire(n,m,num,pref,cul):
  citeşte n
  pentru i=1,n execută:
    citeşte num[i]
  sfârşit pentru
  citeşte m
  pentru i=1,m execută:
    cul[i] ← 0
  sfârşit pentru
  pentru i=1,n execută:
    citeşte pref[i]
    cul[pref[i]] ← cul[pref[i]] + 1
                                  { culoarea pref[i] a apărut încă odată }
  sfârşit pentru
sfârşit subalgoritm
```

În algoritm vom căuta valoarea maximă în acest şir, apoi pe acei copii care au ales culoarea *i*, unde *i* este poziția pe care s-a găsit valoarea maximă. În exemplu, valoarea maximă în tabloul *cul* este egal cu 4 şi se găseşte pe poziția *poz* = 1. Această variabilă astfel reține codul unei culori. Se caută în tabloul *pref* indicii elementelor egale cu 1 (*selectare*). Aceştia sunt: Pop Mihai, Popan Ana, Popa Ion, Popescu Marius.

Acum trebuie să găsim grupul în care sunt mai puțini copii, dar care, după ce am pus primul grup deoparte, sunt cei mai numeroşi. Înseamnă că ne interesează „al doilea maxim" din tabloul *cul*. Pentru a uşura găsirea acestuia, vom înlocui în tabloul *cul* valoarea 4 cu 0. La pasul următor maximul din tabloul *cul* = (0, 1, 2) este 2 pe poziția 3 (adică culoarea 3). În tabloul *pref* se caută toți indicii (copiii) elementelor de valoare *poz*. Aceştia sunt: Rus Andrei, Rusu Teodor.

Se înlocuieşte valoarea 2 cu 0, deci *cul* = (0, 1, 0). Acum *max* se găseşte pe poziția 2 şi este egal cu 1. Se caută, pe baza tabloului *pref* toți copiii care preferă culoarea 2. Aceştia sunt: Trif Ionuț.

Se înlocuieşte valoarea 1 cu 0 în tabloul *cul*, acesta devenind *cul* = (0, 0, 0) şi procesul se încheie.

```
Subalgoritm Afişare:
  pentru k=1,m execută:                    { copiii se grupează în m grupuri }
    Maxim(m,cul,poz)                  { maximul din şirul cul se află pe poziția poz }
    scrie k,' '                                                  { codul culorii }
    pentru i=1,n execută:
      dacă pref[i] = poz atunci { afişăm copiii care preferă culoarea poz }
        scrie num[i],' '
      sfârşit dacă
    sfârşit pentru
    cul[poz] ← 0                                  { eliminăm "ultimul maxim" }
  sfârşit pentru
sfârşit subalgoritm
```