Numere

Capitolul

3

- **Prelucrări ale numerelor**
- **Construirea numărului întreg din caractere cifre Schema lui Horner**
- **Descompunerea unui număr dat în cifre**
- **Implementări sugerate**
- **Probleme propuse**
- **Soluțiile problemelor**

3.1. Prelucrări ale numerelor

Atunci când scriem un număr (cu creionul, sau cu ajutorul tastaturii), aşezăm cifre una lângă alta, dar semnificaţia şirului de cifre este clară pentru toată lumea. Nu ne vom gândi la ceea ce vedem (pe hârtie sau pe monitor) ca la un şir de cifre, ci ca la o singură entitate, adică la un număr. Dacă nu am scris şi virgula zecimală, (sau punctul) avem un număr întreg.

3.1.1. Construirea numărului întreg din caractere cifre Schema lui Horner

Atunci când într-un program se citeşte un număr întreg, au loc câteva operaţii ascunse de ochii utilizatorului. În primul rând cifrele constituie caractere care, imediat după ce au fost citite din fişierul standard sau unul creat de utilizator sunt verificate. „Nimeni nu ştie" de dinainte câte cifre va avea numărul. De asemenea, nimeni nu ştie dacă, din greşeală nu va apărea între două cifre un caracter nepermis. Dar ce anume este nepermis? Caracterele spaţiu, **Tab** şi marca de sfârşit de linie constituie separatori. Altfel spus, apariţia lor semnalează faptul că numărul nu mai are cifre. Dar şi marca de sfârşit de fişier semnalează acelaşi lucru, diferenţa faţă de cele înşirate anterior este că marca de sfârşit de fişier nu joacă rol de „separator", ci de „terminator". Revenind la caracterele permise şi nepermise, se vor respecta regulile sintactice ale limbajului de programare folosit.

În cele ce urmează vom realiza un algoritm pentru citirea unor caractere din care vom crea un număr întreg, dacă acest lucru va fi posibil.

Exemplu

Fie şirul de caractere introdus de la tastatură: 2015600 şi dorim ca această valoare să i se atribuie variabilei *n* de tip întreg, reprezentat pe două cuvinte (în limbajul Pascal, tipul predefinit este `Longint`, în C este `long`). În cele ce urmează vom continua prezentarea conform „posibilităților" din Pascal.

După ce se preia primul caracter încă nu se ştie rangul cifrei corespunzătoare în număr, dar ea trebuie totuşi prelucrată. Pentru a înțelege ideea care poartă denumirea „schema lui *Horner*", vom scrie numărul sub forma unei sume:

$2015600 = 10 \cdot (10 \cdot (10 \cdot (10 \cdot (10 \cdot (10 \cdot 2 + 0) + 1) + 5) + 6) + 0) + 0 =$

$= 2 \cdot 10^6 + 0 \cdot 10^5 + 1 \cdot 10^4 + 5 \cdot 10^3 + 6 \cdot 10^2 + 0 \cdot 10^1 + 0 \cdot 10^0.$

Algoritmul se conturează mai clar din tabelul următor:

cifra curentă	număr în construcție
	0
2	2
0	20
1	201
5	2015
6	20156
0	201560
0	2015600

```
Algoritm Horner_1:
  n ← 0                                   { iniţializarea numărului n }
  cât timp mai există cifre execută:
    citeşte cifra                         { cifra curentă de la stânga la dreapta }
    n ← n*10 + cifra                      { introducerea cifrei în numărul n }
  sfârşit cât timp
  scrie n
sfârşit algoritm
```

Acest algoritm nu este însă suficient de precis, dar mai ales nu este „precaut". În primul rând el se bazează pe faptul că *în mod garantat se citesc cifre*. Deci ar trebui să introducem şi verificarea caracterului pentru a fi siguri că vom prelucra cifre.

A doua observație se referă la condiția din instrucțiunea repetitivă: trebuie să ştim când se termină de fapt şirul de caractere. Şi mai avem o problemă: s-ar putea ca numărul *n* să nu poată fi reprezentat pe spațiul rezervat în memoria calculatorului pentru variabile de tipul lui *n*. În concluzie, trebuie să ne protejăm programul de surprize neplăcute, care ar însemna că un număr pozitiv înmulțit cu 10 să devină în calculator număr negativ (sau pozitiv, dar mai mic) din cauza depăşirii spațiului de memorare alocat. Apariția acestor erori se explică prin faptul că reprezentarea numărului nu mai încape pe pozițiile binare alocate şi un bit semnificativ ajunge pe bitul semn, iar restul

biţilor din stânga se pierd. Dacă pe bitul semn ajunge un bit egal cu 1, vom avea un număr negativ, dacă acesta este 0, numărul este pozitiv, dar din cauza biţilor semnificativi de rang cel mai mare pierduţi din reprezentarea numărului avem un „rezultat" eronat (vezi problema factorial din capitolul 2).

Presupunem că *n* este declarat de tip `Longint` şi este citit caracter după caracter (de tip `Char`) până la marca de sfârşit de linie. Algoritmul îmbunătăţit este următorul:

```
Algoritm Horner_2:
  n ← 0                                          { iniţializarea numărului n }
  prea_mare ← fals
  caracter_străin ← fals
  cât timp nu prea_mare şi nu caracter_străin şi
                              nu urmează marca de sfârşit de linie execută:
    citeşte caracter              { caracterul curent de la stânga la dreapta }
    dacă caracter este cifră atunci
      cifra ← valoarea numerică a caracterului
      dacă n este mai mare decât cel mai mare număr împărţit la 10 atunci
        n ← n*10 + cifra                  { introducerea cifrei în numărul n }
      altfel
        prea_mare ← adevărat
      sfârşit dacă
    altfel
      caracter_străin ← adevărat
    sfârşit dacă
  sfârşit cât timp
  dacă prea_mare atunci
    scrie 'Numar prea mare.'
  altfel
    dacă caracter_străin atunci
      scrie 'Caracter strain.'
    altfel
      scrie n
    sfârşit dacă
  sfârşit dacă
sfârşit algoritm
```

3.1.2. Descompunerea unui număr dat în cifre

Operaţia „inversă", adică furnizarea cifrelor unui număr este necesară ori de câte ori dorim să realizăm o prelucrare cifră cu cifră. Pentru a descompune un număr întreg în cifre vom folosi operaţiile de împărţire întreagă şi determinarea restului împărţirii întregi. Pentru aceste două operaţii în pseudocod vom folosi notaţiile [*m*/*n*] şi *rest*[*m*/*n*].

Exemplificăm algoritmul pentru a determina dacă numărul *n* dat este sau nu palindrom. Un număr *palindrom* are aceeaşi valoare dacă îl „citim” de la dreapta la stânga şi de la stânga la dreapta. Altfel spus, un număr palindrom este egal cu numărul format din cifrele sale luate în ordine inversă (acesta în algoritm este numit *Ninvers*).

```
Algoritm Palindrom:
  citeşte n
  Ninvers ← 0                              { iniţializarea numărului Ninvers }
  CopieN ← n
  cât timp n ≠ 0 execută:                  { cât timp n mai are cifre }
    cifra ← rest[n/10]                     { cifra curentă de la dreapta la stânga }
    n ← [n/10]                             { eliminăm ultima cifră din numărul n }
    Ninvers ← Ninvers*10 + cifra
                                           { construim numărul Ninvers cu schema lui Horner }
  sfârşit cât timp
  dacă Ninvers = CopieN atunci
    scrie 'Palindrom.'
  altfel
    scrie 'Nu e palindrom.'
sfârşit algoritm
```

3.2. Implementări sugerate

Pentru a vă familiariza cu modul în care trebuie rezolvate problemele în cadrul cărora intervin operaţii cu cifre, vă sugerăm să încercaţi să implementaţi algoritmi pentru:

1. afişarea cifrelor unui număr;
2. determinarea numărului de cifre ale unui număr;
3. determinarea celei mai mari (mici) cifre a unui număr;
4. determinarea celei mai mari (mici) cifre a unui număr şi a poziţiilor pe care apare aceasta;
5. determinarea inversului unui număr;
6. generarea unui număr dintr-un şir de caractere care sunt cifre (protecţie în caz de depăşire şi de caractere care nu sunt cifre);
7. verificarea faptului că un număr este palindrom;
8. determinarea tuturor numerelor palindroame dintr-un interval dat;
9. citirea unui număr real de forma *x.y* şi afişarea numărului *y.x*;
10. determinarea tuturor numerelor de tip „vale” dintr-un interval;
11. determinarea tuturor numerelor de tip „munte” dintr-un interval;
12. determinarea tuturor numerelor de tip „vale” şi a tuturor numerelor de tip „munte” dintr-un interval.

3.3. Probleme propuse

3.3.1. Prima zi

Locuitorii din Țara lui Papură Vodă obișnuiau să-şi pună pe căsuțe numere ciudate şi lungi. Piticul Aşchiuță, angajat de probă la Oficiul pentru Evidența Clădirilor Publice, s-a pus pe treabă încă din prima zi, experimentând unele modificări în numerotarea clădirilor. Pentru început el a eliminat cifra 2 din numărul înscris pe clădirea în care lucra.

Scrieți un program care citeşte un număr natural şi elimină din acesta toate aparițiile cifrei 2.

Date de intrare
Din fişierul de intrare **`ZIUA1.IN`** se va citi un număr natural n.

Date de ieşire
În fişierul de ieşire **`ZIUA1.OUT`** se va scrie numărul din care s-au eliminat toate cifrele egale cu 2.

Restricții şi precizări
- $1 \le n \le 1000000000$;
- dacă toate cifrele sunt egale cu 2, rezultatul va fi numărul 0.

Exemplu

`ZIUA1.IN`	**`ZIUA1.OUT`**
`32245`	`345`

3.3.2. A doua zi

A doua zi, piticul Aşchiuță, fiind angajat de probă la Oficiul pentru Evidența Clădirilor Publice, s-a jucat din nou cu cifra lui preferată, numărând de câte ori apare cifra 2 în total în toate numerele caselor de pe strada sa.

Realizați un program care să numere de câte ori apare cifra 2 în toate numerele existente într-un fişier dat.

Date de intrare
În fişierul de intrare **`ZIUA2.IN`** se află numere naturale, aşezate câte unul pe o linie.

Date de ieşire
Numărul de apariții ale cifrei 2 se va scrie în fişierul de ieşire **`ZIUA2.OUT`**.

Exemplu

ZIUA2.IN	ZIUA2.OUT
32245 5514 202 12 7	5

3.3.3. A treia zi

În cea de-a treia zi de lucru, piticul Aşchiuță s-a plictisit şi s-a hotărât să rotească cifrele numerelor caselor tuturor vecinilor spre dreapta. Cunoscând numărul de rotații efectuate asupra cifrelor, precum şi numerele caselor vecinilor de pe stradă, afişați numerele obținute în urma rotațiilor cifrelor.

Date de intrare

Pe prima linie a fişierului **ZIUA3.IN** se află un număr natural *n*, reprezentând numărul de rotații care trebuie efectuate asupra numerelor căsuțelor vecinilor de pe stradă. Pe următoarele linii se află *n* numere naturale, câte un număr pe o linie.

Date de ieşire

În fişierul de ieşire **ZIUA3.OUT** se vor scrie numerele rotite, câte unul pe o linie.

Restricții şi precizări

- $0 \le n \le 255$.

Exemplu

ZIUA3.IN	ZIUA3.OUT
4 32245 567789 223	22453 778956 322

3.3.4. A patra zi

A patra zi piticul Aşchiuță a luat hotărârea de a scurta numerele caselor, deoarece i se păreau prea lungi. El a început să experimenteze ideea de a înlocui fiecare cifră a numărului cu complementul cifrei respective față de cea mai mare cifră din număr, efectuând aceeaşi prelucrare pe rezultatul obținut, el a observat că anumite numere se reduc la o cifră.

Scrieți un program care reduce un număr dat la o cifră cu metoda lui Aşchiuță, dacă acest lucru este posibil.

Date de intrare
Numărul natural *n* se citeşte din fişierul de intrare **ZIUA4.IN**.

Date de ieşire
Cifra rezultată se va scrie în fişierul **ZIUA4.OUT**. În cazul în care numărul nu se poate reduce la o cifră, în fişierul de ieşire se va scrie mesajul: `'imposibil'`.

Restricţii şi precizări
- $1 \le n \le 100000000$ (o sută de milioane).

Exemple

ZIUA4.IN	**ZIUA4.OUT**	
1234	1	Valorile succesive ale numărului până la obţinerea rezultatului sunt: 1234 ⇒ 3210 ⇒ 0123 ⇒ 210 ⇒ 012 ⇒ 10 ⇒ 01 Rezultatul final este 1.
ZIUA4.IN	**ZIUA4.OUT**	
195	4	Valorile succesive ale numărului până la obţinerea rezultatului sunt: 195 ⇒ 804 ⇒ 084 ⇒ 4 Rezultatul final este 4.
ZIUA4.IN	**ZIUA4.OUT**	
29340	imposibil	Valorile succesive ale numărului până la obţinerea rezultatului sunt: 29340 ⇒ 70659 ⇒ 29340 ⇒ … ciclu infinit, acest număr nu se poate reduce la o cifră.

3.3.5. A cincea zi

După cum ştim deja, locuitorii din Ţara lui Papură Vodă obişnuiau să-şi pună pe căsuţe numere foarte ciudate şi lungi. Piticul Aşchiuţă, angajat la Oficiul pentru Evidenţa Clădirilor Publice, în cea de-a cincea zi de lucru s-a apucat de făcut statistici. El a căutat în cazul unui număr *n* cifrele având cea mai mare frecvenţă.

Scrieţi un program care determină frecvenţa maximă de apariţie a unei cifre şi cifrele care au această frecvenţă de apariţie în număr.

Date de intrare
Numărul natural *n* se citeşte din fişierul **ZIUA5.IN**.

Date de ieşire
În fişierul **ZIUA5.OUT** se va scrie frecvenţa maximă, urmată de spaţiu, apoi se va scrie cifra sau cifrele care au această frecvenţă de apariţie în număr.

Restricţii şi precizări
- $1 \le n \le 1000000000$;
- Cifrele având frecvenţa maximă se vor despărţi prin câte o virgulă.

Exemplu

ZIUA5.IN	**ZIUA5.OUT**
3224537	2 2,3

3.3.6. A şasea zi

A şasea zi piticul Aşchiuţă a umblat prin tot regatul pentru a face o statistică privind numărul de case care sunt numerotate cu numere în aspect de ***urcuş***, ***coborâş***, ***deal*** sau ***vale***.

Spunem că un număr are aspect de *deal* dacă cifrele sale se succed întâi crescător şi apoi descrescător şi considerăm un număr ca având aspect de *vale* dacă cifrele sale se succed întâi descrescător şi apoi crescător. Într-un număr cu aspect de *urcuş* cifrele apar în ordine crescătoare, iar într-unul cu aspect de *coborâş*, în ordine descrescătoare.

Să se scrie un program care realizează statistica dorită de Aşchiuţă.

Date de intrare
În fişierul de intrare **ZIUA6.IN** se află mai multe numere naturale.

Date de ieşire
În fişierul **ZIUA6.OUT** statistica rezultată va fi afişată astfel:
- pe prima linie se va scrie cuvântul `urcus`, urmat de caracterul `':'`, apoi de numărul de numere având aspect de urcuş;
- pe a doua linie se va scrie cuvântul `coboras`, urmat de caracterul `':'`, apoi de numărul de numere având aspect de coborâş;
- pe a treia linie se va scrie cuvântul `deal`, urmat de caracterul `':'`, apoi de numărul de numere având aspect de deal;
- pe a patra linie se va scrie cuvântul `vale`, urmat de caracterul `':'`, apoi de numărul de numere având aspect de vale;
- dacă în fişierul de intrare există numere care nu se încadrează în nici una dintre cele patru grupe, pe a cincea linie se va scrie cuvântul `oarecare`, urmat de caracterul `':'`, apoi de numărul de numere corespunzător.

Restricţii şi precizări
- $1 \le$ *număr citit* ≤ 1000000000.

Exemplu

ZIUA6.IN
```
32245 123 765 789 4563 8954
1123 678 131313
```

ZIUA6.OUT
```
urcus:4
coboras:1
deal:2
vale:1
oarecare:1
```

3.3.7. A şaptea zi

În cea de-a şaptea zi lui Aşchiuță îi trecu prin minte că ar fi mai interesant dacă numerele caselor ar fi în următoarea succesiune: 11, 22, 33, 44, 55, 66, 77, 88, 99, 111, 222, 333, …

Să se genereze *n* numere de case conform regulii lui Aşchiuță.

Date de intrare

În fişierul de intrare **ZIUA7.IN** avem scris un număr natural *n*.

Date de ieşire

Numerele generate după regula de mai sus vor fi scrise în fişierul **ZIUA7.OUT**, câte un număr pe fiecare linie.

Restricții şi precizări

- $1 \le n \le 1000$.

Exemplu

ZIUA7.IN
```
5
```

ZIUA7.OUT
```
11
22
33
44
55
```

3.4. Soluțiile problemelor

3.4.1. Prima zi

Fie, de exemplu, numărul 232271. În tabelul următor coloana *nr* conține valoarea curentă a numărului care se prelucrează, coloana *cifra* reține valoarea cifrei analizate, iar *putere* reprezintă valoarea puterii lui 10 în procesul de construire a rezultatului. În această problemă nu vom aplica schema lui Horner, deoarece aplicarea ei ar conduce la un număr în care cifrele ar fi aşezate în ordine inversă. Valoarea variabilei *putere* ne va ajuta ca înmulțind numărul cu puterea lui 10 corespunzătoare, să aşezăm cifra exact pe poziția având rangul dorit.

nr	*cifra*	*putere*	*rezultat*
232271		1	0
232271	**1**	1	1
23227	**7**	10	71
2322	2	10	71
232	2	10	71
23	**3**	100	371
2	2		
0			

Vom descrie algoritmul conform căruia am completat tabelul de variație.

```
Algoritm Ziua1:
   citeşte nr
   rezultat ← 0   { rezultatul va fi un număr nou, care se calculează ca sumă de }
                                                    { termeni; element neutru: 0 }
   putere ← 1 { puterile lui 10 se obțin în urma unor înmulțiri; element neutru: 1 }
   cât timp nr ≠ 0 execută:
     cif ← rest[nr/10]                  { reținem ultima cifră din numărul dat }
     nr ← [nr/10]                        { tăiem ultima cifră din numărul dat }
     dacă cif ≠ 2 atunci { dacă cifra reținută nu este 2, o adăugăm la rezultat }
       rez ← rez + cif*p
       p ← p*10                                  { pregătim p pentru pasul următor }
     sfârşit dacă
   sfârşit cât timp
   scrie rez
sfârşit algoritm
```

3.4.2. A doua zi

Pentru exemplificare vom considera un singur număr şi vom reține acțiunile care trebuie realizate asupra numerelor pentru a număra cifrele egale cu 2 în toate numerele citite.

număr	*cifră*	*contor*
232271	1	0
23227	7	
2322	2	1
232	2	2
23	3	
2	2	3
0		

Rezultatul obținut este 3, reprezentând numărul cifrelor egale cu 2 în total în numărul 232271.

```
Algoritm Ziua2:
  câte ← 0
  cât timp nu urmează marca de sfârşit de fişier execută:
    citeşte n
    cât timp n ≠ 0 execută:
      cif ← rest[n/10]              { reţinem ultima cifră din numărul dat }
      n ← [n/10]                    { „tăiem” ultima cifră din numărul dat }
      dacă cif = 2 atunci                  { dacă cifra este 2, o numărăm }
        câte ← câte + 1
      sfârşit dacă
    sfârşit cât timp
  sfârşit cât timp
  scrie câte
sfârşit algoritm
```

3.4.3. A treia zi

Vom considera un exemplu pe care vom analiza paşii algoritmului de rezolvare. Vrem să rotim numărul 1234567 cu 10 poziţii spre dreapta. Observăm că numărul va fi rotit o dată complet şi cu încă trei poziţii.

Pentru fiecare număr citit trebuie să:

- numărăm câte cifre are numărul (în exemplu: 7 cifre);
- determinăm numărul de rotaţii care „merită” efectuate (observăm că rotaţiile complete nu modifică configuraţia cifrelor numerelor); în funcţie de relaţia dintre *n* (numărul rotaţiilor) şi numărul cifrelor numărului, calculăm numărul de rotaţii astfel încât să fie mai mic decât numărul de cifre ale numărului; observăm că *număr_ rotaţii_de_efectuat* este restul împărţirii lui *n* la numărul de cifre;
- extragem „în bloc” ultimele *număr_rotaţii_de_efectuat* cifre (în exemplu extragem 567, deci reţinem numărul 1234);
- calculăm 10 la puterea *numărul_cifrelor – număr_rotaţii_de_efectuat* pentru a deplasa cifrele de rotit, în faţa numărului (puterea lui 10 de care avem nevoie în exemplul nostru este 10000);
- calculăm suma dintre numărul rămas şi cifrele extrase înmulţite cu puterea lui 10 pregătită; pentru exemplul nostru rezultatul este: $1234 + 567 \cdot 10000 = 5671234$.

Prezentăm algoritmul descris mai sus în pseudocod:

```
Algoritm Ziua3:
  citeşte k                          { numărul rotaţiilor cerute iniţial }
```

```
  cât timp nu urmează marca de sfârşit de fişier execută:
    citeşte n                              { numărul curent în care vom roti cifrele }
    nrcif ← 0        { numărul cifrelor în numărul curent în care vom roti cifrele }
    m ← n                        { copie a lui n, deoarece urmează să-l modificăm }
                              { pe de altă parte mai avem nevoie de valoarea lui n }
    cât timp m ≠ 0 execută:                { numărăm câte cifre are numărul }
      nrcif ← nrcif + 1
      m ← [m/10]                          { câtul împărţirii întregi a lui n la 10 }
    sfârşit cât timp
    kk ← rest[k/nrcif]                { numărul rotaţiilor care trebuie efectuate }
                                                    { de k original mai avem nevoie }
    p ← 1                                  { extragem în bloc ultimele k cifre }
    pentru i=1,kk execută:
      p ← p*10                              { calculăm a kk-a putere a lui 10 }
    sfârşit pentru
    de_mutat ← rest[n/p]
    n ← [n/p]
         { ceea ce a rămas din număr după ce secvenţa „de mutat" a fost extrasă }
    p ← 1                                          { calculăm puterea lui 10 }
    pentru i=1,nrcif-kk execută:
      p ← p*10
    sfârşit pentru
    n ← n + de_mutat*p                        { construim rezultatul în n }
    scrie n
  sfârşit cât timp
sfârşit algoritm
```

3.4.4. A patra zi

Pentru a determina algoritmul de rezolvare vom considera un exemplu. Vom complementa cifrele numărului 122375 față de cifra maximă. Asupra rezultatului aplicăm aceeaşi prelucrare, luând în considerare cifra maximă a numărului obținut la pasul precedent.

Pentru fiecare valoare nouă, vom efectua următoarele operații:

- determinăm cifra de valoare maximă dintre cifrele numărului;
- complementăm cifrele numărului față de această cifră.

În algoritmul propus vom lucra cu următoarele variabile, având semnificațiile precizate:

- *n* este numărul natural dat;
- *m* este o variabilă auxiliară în care vom copia valoarea lui *n*;

- *cif* este cifra curentă;
- *nouln* este noul număr format;
- *cmax* este cifra maximă a numărului;
- *p* este o putere a lui 10 care ne ajută în formarea numărului nou.

n	*m*	*cif*	*nouln*	*cmax*	*p*	*Explicații*
122375	122375	5		5		calcul cifră maximă
	12237	7		7		
	1223	3				
	122	2				
	12	2				
	1	1				
	0					
122375	122375	2	2	7	1	7 – 5 = 2
	12237	0	02		10	7 – 7 = 0
	1223	4	402		100	7 – 3 = 4
	122	5	5402		1000	7 – 2 = 5
	12	5	55402		10000	7 – 2 = 7
	1	6	655402		100000	7 – 1 = 6
	0					
655402	655402	2		2		calcul cifră maximă
	65540					
	6554			4		
	655			5		
	65					
	6			6		
	0					
…						

Se observă că la fiecare complementare am parcurs cifrele unui număr de două ori. Ne punem problema de a face o singură parcurgere. Încercăm deci să determinăm cifra maximă în ciclul de complementare a numărului. Inconvenientul care apare în această situație este că la primul pas nu avem nici o cifră cu care să putem inițializa variabila care păstrează cifra maximă. Deci, fie determinăm prima cifră anterior descompunerii numărului în cifre, fie inițializăm variabila cu 0 (cea mai mică cifră).

Vom folosi două copii ale lui *n* (versiunea mai veche a lui *n* şi versiunea anterioară a acestuia) care ajută la depistarea situației în care transformarea totală a numărului este imposibilă, deoarece s-ar ajunge într-un ciclu infinit. De exemplu, dacă transformăm 102, la primul pas, se obține 120, iar la al doilea din nou 102.

```
Algoritm Ziua4:
  citeşte n
  anterior ← 0
  imposibil ← fals
  cmax ← 0                    { valoare iniţială pentru cifra de valoare maximă }
  m ← n                       { m este copia lui n; (de n mai avem nevoie) }
  cât timp m ≠ 0 execută:     { stabilim cifra de valoare maximă }
    cif ← rest[m/10]          { cifra curentă }
    dacă cif > cmax atunci
      cmax ← cif              { actualizarea cifrei maxime }
    sfârşit dacă
    m ← [m/10]
  sfârşit cât timp
  cât timp (n > 9) şi nu imposibil execută:   { complementare }
    m ← n
    p ← 1
    veche ← anterior          { „veche" este versiunea mai veche a lui n }
    anterior ← n              { „anterior" este versiunea anterioară a lui n }
    n ← 0                     { număr nou }
    nextcmax ← 0
    cât timp m ≠ 0 execută:
      cif ← cmax - rest[m/10]   { complementăm ultima cifră din număr }
      n ← n + cif*p             { construim numărul }
      m ← [m/10]
      p ← p*10                  { următoarea putere a lui 10 }
      dacă cif > nextcmax atunci
        nextcmax ← cif          { actualizarea cifrei maxime pentru pasul următor }
      sfârşit dacă
    sfârşit cât timp
    cmax ← nextcmax
    dacă n = veche atunci
      imposibil ← adevărat      { am intrat în ciclu infinit }
    altfel
      imposibil ← fals
  sfârşit cât timp
  dacă imposibil atunci
    scrie 'imposibil'
  altfel
    scrie n
  sfârşit dacă
sfârşit algoritm
```

3.4.5. A cincea zi

Vom căuta să descoperim algoritmul de rezolvare, pornind de la un exemplu: să se determine cifrele cu număr maxim de apariţii în numărul 23322731. În reprezentarea algoritmului vom folosi următoarele notaţii:

- *nr* este numărul dat;
- *copie* va păstra valoarea numărului pentru a-l putea studia în cazul fiecărei cifre *cif*;
- *cif* va fi cifra a cărei frecvenţă dorim să o determinăm;
- *cifcurentă* reprezintă cifra din *nr* cu care vom compara cifra *cif*;
- *aparcurent* reprezintă numărul de apariţii ale cifrei *cif* în *nr*;
- *aparmax* reprezintă numărul maxim de apariţii.

Analizând posibilităţile de a număra apariţiile unei cifre, observăm că atâta timp cât numărul mai are cifre, va trebui să desprindem din număr ultima cifră şi va trebui să o comparăm cu valoarea cifrei *cif*, generată cu **`pentru`**. Dacă cele două coincid, mărim contorul *aparcurent*. Astfel, parcurgând cifrele numărului, putem contoriza fiecare cifră între 0 şi 9. De asemenea, se poate actualiza şi valoarea contorului maxim (*aparmax*). Dar ne izbim de următoarea problemă: în enunţ ni se cere să scriem în fişier ***toate*** cifrele având frecvenţa de apariţie maximă. Deci, să presupunem că am găsit deja în *nr* doi de 0, dar nu ştim dacă trebuie să scriem această valoare în fişier sau nu, deoarece nu ştim dacă, de exemplu, din cifra 1 vom găsi mai multe apariţii sau nu. Soluţia este simplă: vom scrie în fişier valoarea presupusă maximă (2) şi cifra 0. Dacă cifra 1 apare o singură dată, nu modificăm nimic. Dacă numărul ei de apariţie este egal cu 2, deschidem fişierul pentru adăugare şi îl scriem şi pe 1. Dacă însă numărul de apariţie este mai mare decât 2, deschidem fişierul pentru scriere (în Pascal cu `ReWrite`), astfel anulând conţinutul lui vechi şi scriem numărul maxim de apariţii împreună cu cifra corespunzătoare.

```
Algoritm Ziua5:
  citeşte nr                                          { numărul dat }
  copie ← nr                                          { copie a lui nr }
  aparmax ← 0                          { iniţializăm frecvenţa de apariţie maximă }
  pentru cif=0,9 execută:                      { generăm cifrele posibile în cif }
    aparcurent ← 0                              { iniţializăm frecvenţa cifrei cif }
    cât timp nr ≠ 0 execută:                    { cât timp în nr mai avem cifre }
      cifcurentă ← rest[nr/10]                        { ultima cifră din nr }
      nr ← [nr/10]                                    { o tăiem din nr }
      dacă cifcurentă = cif atunci  { dacă cifra curentă este egală cu cif }
        aparcurent ← aparcurent + 1          { creştem frecvenţa cifrei cif }
      sfârşit dacă
    sfârşit cât timp                        { am parcurs cifrele numărului nr }
```

```
    dacă aparmax < aparcurent atunci          { dacă avem un maxim nou }
      aparmax ← aparcurent                        { actualizăm maximul }
      deschidem fişierul; dacă a avut conţinut (un maxim mai mic) acesta se pierde
      scrie aparmax,' ',cif                  { scriem maximul şi cifra cif }
    altfel
      dacă aparmax = aparcurent atunci       { avem o frecvenţă egală cu }
                { aparmax care deja este scris în fişier, împreună cu o valoare cif }
        deschidem fişierul pentru adăugare
        scrie ',',cif                { scriem cifra cif, având aceeaşi frecvenţă }
      sfârşit dacă
    sfârşit dacă
    nr ← copie      { refacem nr pentru a putea căuta în el următoarea cifră cif }
  sfârşit pentru
sfârşit algoritm
```

3.4.6. A şasea zi

Ideea de rezolvare a acestei probleme este de a număra numărul schimbărilor în sensul ordonării şirului cifrelor numărului.

Vom păstra într-o variabilă *d* direcţia iniţială (cresc cifrele sau descresc?) a ordonării cifrelor numărului. Deoarece ne este mai uşor să analizăm ultima cifră din număr, această direcţie va fi calculată din *diferenţa dintre ultima şi penultima cifră a numărului dat*. Deci *d* se iniţializează cu `rest[n/10] - rest[[n/10]/10]`.

În continuare ne va interesa doar semnul (+ sau –) al variabilei *d*.

- Dacă în final numărul schimbărilor direcţiei este egal cu 0 şi dacă $d > 0$, atunci avem un număr cu aspect de *urcuş* (exemplu: $n = 1259$, $d = 9 – 5 = 4 > 0$), iar dacă $d < 0$, atunci avem *coborâş* (exemplu: $n = 6542$, $d = 2 – 4 = –2 < 0$).
- Dacă $d = 0$, cifrele sunt identice, deci trebuie să calculăm *d* pe baza primelor cifre distincte ale numărului.
- Dacă numărul schimbărilor direcţiei este egal cu 1 şi $d > 0$, atunci avem un număr cu aspect de *vale* (exemplu: $n = 76345$, $5 – 4 = 1 > 0$), iar dacă $d < 0$, atunci avem un *deal* (exemplu: $n = 1273$, $3 – 7 = –7 < 0$).
- Numerele care nu intră în nici una dintre aceste categorii sunt considerate *oarecare* (de exemplu: 131313).

```
Algoritm Ziua6:
  nrurc ← 0
  nrcob ← 0
  nrdeal ← 0
  nrvale ← 0
  nroarecare ← 0                          { iniţializarea contoarelor }
```

```
cât timp nu urmează marcajul de sfârşit de fişier execută:
  citeşte n
                { cât timp ultimele două cifre sunt identice, eliminăm ultima cifră }
  cât timp (n > 9) şi (rest[n/10] = rest[[n/10]/10]) execută:
    n ← [n/10]
  sfârşit cât timp
  k ← 0                                   { numărul schimbărilor de direcţie }
  d ← 0                                                        { direcţia }
  dacă n > 9 atunci { dacă numărul rămas este format măcar din două cifre }
    dacă rest[n/10] < rest[[n/10]/10] atunci  { iniţializăm direcţia }
      d ← -1
    altfel   { nu pot fi egale, deoarece am eliminat cifrele egale de la început }
      d ← 1
    sfârşit dacă
    n ← [n/10]                                  { renunţăm la ultima cifră }
    cât timp n > 9 execută:    { cât timp mai există cel puţin 2 cifre în n }
     { dacă direcţia curentă şi diferenţa ultimelor două cifre nu au acelaşi semn }
      dacă ((d < 0) şi (rest[n/10] > rest[[n/10]/10]) sau
           ((d > 0) şi (rest[n/10] < rest[[n/10]/10]) atunci
        k ← k + 1                                { schimbare de direcţie }
        d ← -d
      sfârşit dacă
      n ← [n/10]
    sfârşit cât timp
  sfârşit dacă
  dacă k = 0 atunci
    dacă d = 0 atunci nroarecare ← nroarecare + 1 { k = 0, d = 0 }
    altfel
      dacă d > 0 atunci nrurc ← nrurc + 1          { k = 0, d > 0 }
                 altfel nrcob ← nrcob + 1          { k = 0, d < 0 }
      sfârşit dacă
    sfârşit dacă
  altfel
    dacă k = 1 atunci
      dacă d < 0 atunci nrvale ← nrvale + 1        { k = 1, d < 0 }
                 altfel nrdeal ← nrdeal + 1        { k = 1, d > 0 }
      sfârşit dacă
    altfel
      dacă k > 1 atunci nroarecare ← nroarecare + 1     { k > 1 }
      sfârşit dacă
    sfârşit dacă
  sfârşit dacă
sfârşit cât timp
```

```
  scrie 'urcus:',nrurc                                    { afişare }
  scrie 'coboras:',nrcob
  scrie 'deal:',nrdeal
  scrie 'vale:',nrvale
  scrie 'oarecare:',nroarecare
sfârşit algoritm
```

3.4.7. A şaptea zi

Enunţul problemei ne cere să generăm primii *n* termeni ai şirului: 11, 22, 33, .., 99, 111, 222, 333, …999, 1111…

Să analizăm câteva modalităţi de abordare a problemei. Primii doi algoritmi pot fi utilizaţi pentru $n \leq 73$ (caz în care ultimul termen în şir este numărul format din 10 cifre egale cu 1).

- O idee ar fi să formăm numerele din cifre: 11 din 1 şi 1, 22 din 2 şi 2.

```
Algoritm Ziua7_1:
  citeşte n
  cif ← 1
  nr_cifre ← 2
  pentru i=1,n execută:      { se formează n numere din nr_cifre egale cu cif }
    număr ← cif
    pentru j=2,nr_cifre execută:
      număr ← număr*10 + cif
    sfârşit pentru
    scrie număr
    dacă cif < 9 atunci      { pregătim cifra şi nr_cifre pentru pasul următor }
      cif ← cif + 1
    altfel
      cif ← 1
      nrcif ← nr_cifre + 1
    sfârşit dacă
  sfârşit pentru
sfârşit algoritm
```

- O altă idee, mai simplă, este de a aduna la fiecare număr format din două cifre valoarea 11, la fiecare număr format din trei cifre valoarea 111 şi aşa mai departe. În acest caz şirul ar putea fi privit astfel: **11**, 11+11=**22**, 22+11=**33**, …

  Variabila *de_adunat* va avea succesiv valoarea 11, apoi 111 şi aşa mai departe.

```
Algoritm Ziua7_2:
  citeşte n
  de_adunat ← 11
  număr ← 11
  pentru i=1,n execută:
    scrie număr
                  { pregătim valoarea variabilei de_adunat pentru pasul următor }
    număr ← număr + de_adunat
    dacă rest[număr/10] = 0 atunci
                        { dacă avem cazul 99 + 11 = 110, sau 999 + 111 = 1110 }
      număr ← număr + 1                                    { 110 + 1 = 111 }
      de_adunat ← de_adunat*10 + 1
    sfârşit dacă
  sfârşit pentru
sfârşit algoritm
```

- Dacă se doreşte rezolvarea acestei probleme pentru $n > 73$, vom scrie în fişier, pur şi simplu, cifre alăturate. Numărul cifrelor dintr-un număr se generează conform regulii generării numerelor din şir.

```
Algoritm Ziua7_2:
  citeşte n
  lungime ← 2                                  { lungimea iniţială a numerelor }
  c ← '1'                                          { prima cifră este 1 }
  pentru i=1,n execută:                        { generăm n termini în şir }
    pentru j=1,lungime execută:       { tipărirea elementului curent din şir }
      scrie c
    sfârşit pentru
    dacă c = '9' atunci
       c ← '1'        { urmează din nou un număr format din cifre egale cu 1 }
       lungime ← lungime + 1          { creşte numărul de cifre în număr }
    altfel
      c ← c + 1                                    { următoarea cifră }
    sfârşit dacă
  sfârşit pentru
sfârşit algoritm
```