Aritmetica numerelor mari

Capitolul 11

- **Calcularea valorii 2^n**
- **Adunarea**
- **Scăderea**
- **Înmulțirea**
- **Probleme propuse**
- **Soluțiile problemelor**

În problemele de concurs pot să apară cerințe care impun prelucrarea anumitor numere întregi „mari” care nu pot fi reprezentate cu toate cifrele lor exacte, folosind tipurile întregi cunoscute de limbajele de programare. Reprezentarea lor ca reale nu este o soluție, deoarece nici în tipurile reale nu putem lucra cu un număr mare de cifre exacte. Dacă trebuie să efectuăm calcule cu aceste numere şi cerințele sunt de aşa manieră încât trebuie determinate toate cifrele exacte ale numărului, trebuie să implementăm „aritmetica numerelor mari”.

11.1. Calcularea valorii 2^n

Vom prezenta această tehnică în cazul determinării valorii lui 2^n, unde n ($n \leq 1000$) este număr natural pozitiv.

Exemplu
$n = 100$, $2^n = 1267650600228229401496703205376$

În cele ce urmează, pentru a exemplifica tehnica de implementare a operațiilor de acest fel vom prezenta modul în care se simulează calcularea lui 2^7.

Ideea este de a lucra pe cifre şi de a ține evidența transportului obținut în urma efectuării operației. Rezultatele parțiale ale efectuării operațiilor pe cifre le vom reține într-un tablou (*rez*).

k	1	2	3	4	5	6	7
2^k	2	4	8	16	32	64	128
rez	(2)	(4)	(8)	(6, 1)	(2, 3)	(4, 6)	(8, 2, 1)

Se observă că în tabloul *rez* am adăugat elemente pe măsură ce s-au realizat înmulțirile cu 2. Pentru a afişa valoarea 2^n vom parcurge tabloul *rez* de la dreapta spre stânga. Deoarece la fiecare pas se generează rezultatul prin înmulțirea cifră cu cifră a rezultatului obținut la pasul anterior cu 2, timpul de lucru este mic, mai ales, dacă (în Pascal) în loc de înmulțire efectivă, lucrăm cu operatorul **shl**.

Să presupunem că s-a realizat calculul lui 2^{k-1} şi tabloul *rez* are *dim* termeni. Urmează să efectuăm o nouă înmulțire cu 2 pentru a-l obține pe 2^k. Pentru fiecare element din şirul rezultat, rez_i, (i = 1, 2, ..., *dim*) efectuăm înmulțirea cu 2. Dacă rezultatul înmulțirii este mai mare decât 10, atunci în rez_i se reține doar cifra unităților, iar cifra zecilor (transportul) se depune în altă variabilă (*rest*), urmând să fie adunată la termenul următor.

Dacă după efectuarea celor *dim* înmulțiri variabila *rest* este diferită de 0, (ultima înmulțire a generat cifră de transport), atunci se mai adaugă un al *dim* + 1-lea element în tablou.

Algoritmul care realizează această înmulțire repetată este următorul :

```
Subalgoritm Înmulțire(exponent,rez):
  putere ← 0
  dim ← 1                                                    { avem o cifră }
  rez[dim] ← 1                                               { aceasta este 1 }
  repetă
    rest ← 0          { în rest vom determina la fiecare pas cifra de transport }
    pentru i=1,dim execută
      nou ← rez[i]*2 + rest                  { calculăm cifra curentă din rez }
                  { în loc de rez[i]*2 în Pascal putem folosi rez[i] shl 1 }
      rest ← nou div 10                                  { calculăm transportul }
      rez[i] ← nou mod 10                          { şi cifra curentă din rez }
    sfârşit pentru
    dacă rest ≠ 0 atunci
              { dacă există cifră de transport, în rez ne trebuie încă un element }
      dim ← dim + 1
      rez[dim] ← rest                { în acest element punem cifra de transport }
    sfârşit dacă
    putere ← putere + 1          { am efectuat o înmulțire cu 2, putere creşte }
                          { până când se atinge valoarea dată în exponent }
  până când putere = exponent
sfârşit subalgoritm
```

Afişarea rezultatului în fişierul de ieşire se face tipărind şirul *rez* de dimensiune *dim* în ordine inversă.

11.2. Probleme propuse

11.2.1. Adunarea numerelor mari

Să se realizeze adunarea a două numere întregi mari care au mai mult de 10 cifre fiecare.

Date de intrare
Valorile celor două numere se citesc din fişierul **NUM.IN** de pe două linii distincte.

Date de ieşire
Rezultatul adunării se va scrie în fişierul **NUM.OUT**.

Restricţii şi precizări
- numerele au cel mult 100 de cifre.

Exemplu

NUM.IN	NUM.OUT
1234567890998 2345677	1234570236675

11.2.2. Scăderea numerelor mari

Să se realizeze scăderea a două numere întregi mari care au mai mult de 10 cifre fiecare.

Date de intrare
Valorile celor două numere se citesc din fişierul **NUM.IN** de pe două linii distincte.

Date de ieşire
Rezultatul scăderii se va scrie în fişierul **NUM.OUT**.

Restricţii şi precizări
- numerele au cel mult 100 de cifre.

Exemplu

NUM.IN	NUM.OUT
1234567890998 2345677	1234565545321

11.2.3. Înmulţirea numerelor mari

Să se realizeze înmulțirea a două numere întregi mari care au mai mult de 10 cifre fiecare.

Date de intrare
Valorile celor două numere se citesc din fişierul **NUM.IN** de pe două linii distincte.

Date de ieşire
Rezultatul înmulțirii se va scrie în fişierul **NUM.OUT**.

Restricţii şi precizări
- numerele au cel mult 100 de cifre.

Exemplu

NUM.IN	**NUM.OUT**
1234 234	288756

11.3. Soluțiile problemelor propuse

11.3.1. Adunarea numerelor mari

Prezentăm un exemplu de adunare a două numere mari. Citim numerele în două şiruri de caractere, apoi le convertim în tablouri de cifre nr_1 şi nr_2. Rezultatul adunării se va memora în tabloul de cifre nr_3.

De la început considerăm ambele tablouri având câte un element 0 în plus pe poziția 0, deoarece, dacă cele două numere au acelaşi număr de cifre, s-ar putea ca prin adunare să rezulte un număr având cu una mai multe cifre decât cele două numere date.

nr_1 = (0, 1, 2, 9, 8, 7, 9), n = 6 cifre (0 din față este pe poziția 0)
nr_2 = (0, 9, 2, 9, 8), m = 4 cifre

În scopul adunării, elementele tabloului nr_2 se mută pe ultimele m poziții considerate pe o lungime egală cu $n + 1$, (presupunând că $n \geq m$), iar pozițiile rămase libere în fața cifrelor numărului se completează cu zerouri.
nr_2 = (0, 0, 0, 9, 2, 9, 8)
nr_3 va conține cifrele sumei; este inițializat cu 0: nr_3 = (0, 0, 0, 0, 0, 0, 0).

Adunarea propriu-zisă o efectuăm cifră cu cifră, pornind de la cifra unităților:
0129879+
0009298

nr_1	0	1	2	9	8	7	9
nr_2	0	0	0	9	2	9	8
nr_3	0	0	0	0	0	0	0

La primul pas: 9 + 8 + 0 = **17**, dar în nr_3 nu vom putea avea decât elemente având valoare mai mică decât 10. Reținem în elementul de pe poziția curentă restul împărțirii întregi la 10 a acestei sume, iar în elementul precedent cifra de transport 1.

nr_1	0	1	2	9	8	7	**9**
nr_2	0	0	0	9	2	9	**8**
nr_3	0	0	0	0	0	**1**	**7**

Urmează adunarea 7 + 9 + 1 = 17:

nr_1	0	1	2	9	8	**7**	9
nr_2	0	0	0	9	2	**9**	8
nr_3	0	0	0	0	**1**	**7**	7

8 + 2 + 1 = 11:

nr_1	0	1	2	9	**8**	7	9
nr_2	0	0	0	9	**2**	9	8
nr_3	0	0	0	**1**	**1**	7	7

9 + 9 + 1 = 19:

nr_1	0	1	2	**9**	8	7	9
nr_2	0	0	0	**9**	2	9	8
nr_3	0	0	**1**	**9**	1	7	7

2 + 0 + 1 = 3:

nr_1	0	1	**2**	9	8	7	9
nr_2	0	0	**0**	9	2	9	8
nr_3	0	**0**	**3**	9	1	7	7

1 + 0 + 0 = 1:

nr_1	0	**1**	2	9	8	7	9
nr_2	0	**0**	0	9	2	9	8
nr_3	**0**	**1**	3	9	1	7	7

Numărul rezultat (suma celor două numere date) este 139177.

În exemplul de mai sus $n > m$. Deoarece este posibil să avem $m > n$, în funcție de numărul de cifre, eventual, vom interschimba nr_1 cu nr_2.

Subalgoritmul care citeşte şirurile de caractere şi creează şirurile de cifre nr_1 şi nr_2:

```
Subalgoritm Citire(nr1,nr2,n,m):          { citim numerele în variabile }
  citeşte a                  { de tip string, apoi le transformăm în şir de cifre }
  n ← lungimea şirului de caractere a          { primul număr are n cifre }
  nr1[0] ← 0                                          { cifra de rang n }
  pentru i=1,n execută:
    nr1[i] ← valoarea numerică a caracterului de indice i
  sfârşit pentru
  citeşte b
  m ← lungimea şirului de caractere b          { al doilea număr are m cifre }
  nr2[0] ← 0                                          { cifra de rang m }
  pentru i=1,m execută:
    nr2[i] ← valoarea numerică a caracterului de indice i
  sfârşit pentru
sfârşit subalgoritm
```

Subalgoritmul care pregăteşte numerele pentru adunarea cifră cu cifră asigură ca primul număr să fie cel care are mai multe cifre, iar pe al doilea îl aranjează astfel încât în procesul de adunare să se adune cifre de rang egal. În acest scop, cifrele numărului având mai puține cifre se mută pe lungimea celui cu mai multe, aliniate la dreapta, iar pozițiile rămase libere se completează cu 0.

```
Subalgoritm Pregătire(n,nr1,nr2):
                    { dacă nr1 are mai puține cifre decât nr2, le interschimbăm }
  dacă n < m atunci
    aux1 ← nr1; aux2 ← n
    nr1 ← nr2; n ← m
    nr2 ← aux1; m ← aux2
  sfârşit dacă
  aux ← n - m
                        { aşezăm cifrele sale pe poziții corespunzătoare cifrelor }
  pentru i=n,aux+1 execută:
    nr2[i] ← nr2[i-aux]                    { din nr1 cu care se vor aduna }
  sfârşit pentru
  pentru i=0,aux execută:                { restul cifrelor se completează cu 0 }
    nr2[i] ← 0
  sfârşit pentru
sfârşit subalgoritm
```

Algoritmul care realizează adunarea este:

```
Subalgoritm Adun(nr1,nr2,nr3):
  pentru i=0,n execută:
    nr3[i] ← 0                  { inițial, cifrele rezultatului sunt egale cu 0 }
  sfârşit pentru
                        { adunăm cifrele de rang 0,1,...,n-1 din cele trei numere }
  pentru i=n,1 execută:
    s ← nr1[i] + nr2[i] + nr3[i]
    dacă s < 10 atunci          { dacă suma cifrelor este din intervalul [0..9] }
      nr3[i] ← s           { cifra corespunzătoare din nr3 va fi această sumă }
    altfel { în caz contrar cifra precedentă primeşte valoarea cifrei de transport }
      nr3[i-1] ← 1                        { când i=1, avem ultimul transport }
                    { cifra curentă primeşte restul împărțirii întregi a sumei la 10 }
      nr3[i] ← s mod 10
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

Afişarea rezultatului se reduce la scrierea elementelor şirului nr_3 în fişier. Singura sarcină care mai trebuie îndeplinită este de a verifica cifra de pe poziţia 0 în nr_3. Dacă aceasta este egală cu 0, adică la ultima adunare nu am avut transport diferit de 0, nu o scriem în fişier.

11.3.2. Scăderea numerelor mari

Prezentăm un exemplu de scădere a două numere. Citim numerele în două şiruri de caractere, apoi le convertim în tablouri de cifre nr_1 şi nr_2. Rezultatul scăderii se va memora în tabloul de cifre nr_3.

Exemplu

Tabloul nr_1 are $n = 6$ elemente: (1, 2, 9, 8, 7, 9).

Tabloul nr_2 are $m = 4$ elemente: (9, 2, 9, 8).

Tabloul nr_2 se completează cu zerouri la început ca să aibă aceeaşi dimensiune ca tabloul nr_1: (0, 0, 9, 2, 9, 8). Tabloul nr_3 este iniţializat cu 0.

Scăderea se realizează conform următorilor paşi:

nr_1	1	2	9	8	7	9
nr_2	0	0	9	2	9	8
nr_3	0	0	0	0	0	0

La primul pas 9 – 8 = **1**:

nr_1	1	2	9	8	7	**9**
nr_2	0	0	9	2	9	**8**
nr_3	0	0	0	0	0	**1**

La al doilea pas ar trebui să scădem 9 din 7. Adăugăm 10 la 7, iar din 8 (de pe poziţia sutelor) scădem 1. Scădem 9 din 17, obţinând 8.

nr_1	1	2	9	**7**	**17**	9
nr_2	0	0	9	2	**9**	8
nr_3	0	0	0	0	**8**	1

Scădem 2 din 7, obţinem 5:

nr_1	1	2	9	**7**	17	9
nr_2	0	0	9	**2**	9	8
nr_3	0	0	0	**5**	8	1

Acum scădem 9 din 9:

nr_1	1	2	**9**	**7**	17	9
nr_2	0	0	**9**	**2**	9	8
nr_3	0	0	**0**	**5**	8	1

Urmează 2 – 0:

nr_1	1	**2**	9	7	17	9
nr_2	0	**0**	9	2	9	8
nr_3	0	**2**	0	5	8	1

Ultima scădere: 1 – 0 = 1.

nr_1	**1**	2	9	7	17	9
nr_2	**0**	0	9	2	9	8
nr_3	**1**	2	0	5	8	1

Rezultatul obţinut este 120581.

În program, asemănător adunării, numărul format din mai puţine cifre decât celălalt se va completa cu cifre de 0.

Algoritmul care realizează această completare cu zerouri este similar cu cel folosit în programul care realiza adunarea:

```
Subalgoritm Preg(n,m,nr2);
  aux ← n - m
  pentru i=n,aux+1 execută:
    nr2[i] ← nr2[i-aux]
  sfârşit pentru
  m ← n
  pentru i=1,aux execută
    nr2[i] ← 0
  sfârşit pentru
sfârşit subalgoritm
```

Algoritmul de mai sus se aplică după ce se stabileşte care tablou trebuie completat cu zerouri. Această decizie se ia în algoritmul de mai jos.

```
Subalgoritm Calc(n,m,nr1,nr2):
  dacă n ≥ m atunci
    Preg(n,m,nr2)
    Scad(nr1,nr2,nr3,n)
  altfel
            { dacă al doilea număr este mai mare decât primul, el va fi descăzutul }
    Preg(m,n,nr1)
    d ←nr1; nr1 ← nr2; nr2 ←d
    k ← n; n ← m; m ← k
    Scad(nr1,nr2,nr3,n)
  sfârşit dacă
sfârşit subalgoritm
```

Algoritmul care realizează scăderea este:

```
Subalgoritm Scad(nr1,nr2,nr3,n):
  pentru i=n,1 execută:
    dacă nr1[i] < nr2[i] atunci
      nr1[i] ← nr1[i] + 10
      nr3[i] ← nr1[i] - nr2[i]
      nr1[i-1] ← nr1[i-1]-1
    altfel
      nr3[i] ← nr1[i] - nr2[i]
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

La fiecare scădere verificăm dacă valoarea descăzutului este mai mare decât valoarea scăzătorului. În caz afirmativ realizăm scăderea şi reţinem rezultatul în cifra corespunzătoare a lui nr_3. În caz contrar împrumutăm 10 de la elementul din stânga, îl decrementăm pe acesta şi efectuăm scăderea.

Afişarea în fişierul de ieşire se face în subalgoritmul de mai jos.

```
Subalgoritm Afişare(nr3):
  i ← 1
  cât timp nr3[i] = 0 execută:        { 0-rile de la început nu le vom afişa }
    i ← i + 1
  sfârşit cât timp
  dacă i > n atunci
                { dacă numerele sunt egale, tabloul nr3 conţine numai valori nule }
  scrie '0'                                        { se va scrie doar un singur 0 }
  altfel
    cât timp i ≤ n execută:
      scrie c[i]
      i ← i + 1
    sfârşit cât timp
  sfârşit dacă
sfârşit subalgoritm
```

11.3.3. Înmulţirea numerelor mari

Prezentăm un exemplu de înmulţire a două numere. Vom reţine cele două numere în tablourile de cifre nr_1 şi nr_2. Rezultatul se va memora în tabloul de cifre nr_3.

Fie nr_1 = 182 şi nr_2 = 87. Produsul lor este 15834.

Număr	*Cifra de rang* 0	*Cifra de rang* 1	*Cifra de rang* 2	*Cifra de rang* 3	*Cifra de rang* 4	*Explicaţii*
nr_1	1	8	2			cifrele primului număr
nr_2		8	7			cifrele celui de-al doilea
	14	56	7			rezultatul parţial după înmulţirea cu cifra 7
		16	64	8		rezultatul parţial după înmulţirea cu cifra 8
nr_3	14	72	71	8		rezultatul înmulţirii după adunarea rezultatelor parţiale

Urmează prelucrarea „cifrelor” rezultatului. Pentru fiecare element mai mare decât 9 determinăm restul împărţirii acestuia la 10 (cifra care va înlocui numărul respectiv) şi câtul care reprezintă transportul care se va aduna la următorul număr din şir.

Număr	*Cifra de rang* 0	*Cifra de rang* 1	*Cifra de rang* 2	*Cifra de rang* 3	*Cifra de rang* 4
nr_3	[14/10] = 1 rest[14/10] = 4	72 + 1 = 73 [73/10] = 7 rest[73/10] = 3	71 + 7 = 78 [78/10] = 7 rest[78/10] = 8	8 + 7 = 15 [15/10] = 1 rest[15/10] = 5	1
nr_3	**1**	**3**	**8**	**5**	**1**

Rezultatul înmulţirii este 15834.

Se observă că se realizează înmulţirea celor două numere cifră cu cifră ca în aritmetică. Rezultatele înmulţirii (pot fi numere formate din una sau două cifre) se păstrează în tabloul `nr3` în ordine inversă, adică de la stânga la dreapta şi fără să se realizeze transferul cifrei zecilor la fiecare pas. Aceste rezultate parţiale se vor păstra în tabloul `nr3` astfel încât rezultatele înmulţirilor curente se vor aduna imediat la elementul anterior, deplasându-ne cu o poziţie la dreapta la fiecare înmulţire cu o cifră nouă din înmulţitor.

La sfârşitul adunărilor prelucrăm tabloul `nr3` astfel încât acesta să conţină cifre: pornim de la stânga la dreapta; în elementul curent păstrăm restul împărţirii întregi la 10 a acestuia, iar partea întreagă a împărţirii la 10 adunăm la elementul următor.

Algoritmul care realizează această înmulţire este:

```
Subalgoritm Înmulţire(nr1,nr2,nr3):
  k ← 0          { k va conţine valoarea cu care se deplasează rezultatele parţiale }
                  { în funcţie de numărul de ordine al cifrei curente din înmulţitor }
  pentru j=m,1 execută:
    pentru i=n,1 execută:
                                                        { calcularea produsului }
      nr3[n-i+1+k] ← nr3[n-i+1+k] + nr1[i]*nr2[j]
    sfârşit pentru
                { după ce s-a înmulţit întreg şirul cu o cifră, se trece la următoarea }
   { rezultatul care s-ar scrie sub cel anterior este deplasat spre dreapta cu o poziţie }
    k ← k + 1
  sfârşit pentru
sfârşit subalgoritm
```

Rezultatul din tabloul `nr3` se prelucrează, deoarece conţine numere şi nu cifre:

```
Subalgoritm Prelucrare(nr3):
  pentru i=1,n+m-2 execută:
    dacă nr3[i] div 10 ≤ atunci
      nr3[i+1] ← nr3[i+1] + (nr3[i] div 10)
      nr3[i] ← nr3[i] mod 10
    sfârşit dacă
  sfârşit pentru
sfârşit subalgoritm
```

Şirul cifrelor se afişează în ordine inversă:

```
Subalgoritm Afişare(nr3):
  pentru i←n+m-1,1 execută:
    scrie nr3[i]
  sfârşit pentru
sfârşit subalgoritm
```