Parcurgerea cifrelor unui număr

Avem o valoare naturală și se pune problema accesării cifrelor sale, ca valori individuale. De exemplu, având o variabilă `n`, de tip int care memorează valoarea `2019`, expresia `n%10 * (n%10)` are ca valoare `81` adică pătratul ultimei cifre a lui `n`. Din exemplul de mai sus se deduce că ne este la îndemână accesarea ultimei cifre întrucât limbajul ne pune la dispoziție operatorul `%` și, cu o expresie de genul `n%10` avem acces direct la acea cifră. Nu același lucru putem spune dacă dorim accesarea primei cifre a numărului memorat. Acest lucru este totuși posibil dacă am cunoaște numărul de cifre ale lui `n`. De exemplu, dacă `n` are `4` cifre am putea scrie `n%1000` și am avea, de asemenea, acces direct la prima cifră.

Pe cazul general însă, când `n` este o valoare citită de la tastatură (care poate avea oricâte cifre), întotdeauna `n%10` dă ultima cifră, dar pentru prima cifră nu dispunem de o expresie.

Este bine de avut în vedere observația mai generală: Dacă $z = 10^k$, atunci:

- expresia `n%z` are ca valoare numărul format cu ultimele `k` cifre ale lui `n`
- expresia `n/z` are ca valoare numărul format fără ultimele `k` cifre ale lui `n`

De exemplu, pentru `n = 1768903`, `z = 1000` avem:

n	z	n%z	n/z
1768903	1000	903	1768

Se pune problema traversării cifră cu cifră pentru valoarea memorată în variabila `n`. Acest lucru este, de exemplu, util pentru a afla câte cifre are `n`.

Așa cum spuneam, prin expresia `n%10` avem acces la ultima cifră, apoi cu o atribuire de forma `n = n/10` facem ca valoarea lui `n` să se modifice încât să "dispară" ultima cifră.

Așadar, executând repetat:

- accesează ultima cifră
- elimină ultima cifră

facem ca una câte una să fie obținute toate cifrele numărului, ca valori individuale, începând de la cea din dreapta. Aceste două instrucțiuni trebuie repetate până se termină cifrele lui `n`. Se ajunge la următorul pseudocod:

```
cât timp n are cifre
      accesează ultima cifră
      elimină ultima cifră
```

Acest cod, tradus în C/C++ este:

```
while (n!=0) {
    operație cu n%10
    n = n/10;
}
```

De exemplu, pentru n = 2167 avem:

- operăm cu `n%10`, adică `7`, apoi prin `n = n/10`, `n` devine `216`
- operăm cu `n%10` adică `6`, apoi prin `n = n/10`, `n` devine `21`
- operăm cu `n%10`, adică `1`, apoi prin `n = n/10`, `n` devine `2`
- operăm cu `n%10` adică `2`, apoi prin `n = n/10`, `n` devine `0`, moment în care ne oprim.

Condiția de oprire, `n!=0`, are semnificația "`n` nu mai are cifre" pentru codul de mai sus, însă în anumite cazuri, când vorbim chiar despre valoarea `0` a lui `n`, putem avea de tratat cazuri particulare, considerând pe `n` ca un număr de o cifră.

Aplicații

1. **Numărul de cifre.** Citim o valoare naturală, într-o variabilă de tip int. Determinați numărul de cifre ale sale.

```
cin>>n;
s = 0;
while(n != 0){
    s = s + 1;
    n = n/10;
}
cout<<s;
```

În codul de mai sus, operația pe care o facem în contul ultimei cifre este mărirea cu `1` a unui contor. Totuși, secvența nu dă rezultatul corect dacă valoarea inițială a lui `n` este `0`. Nu s-ar intra în while iar `s` ar rămâne `0`. În acest caz însă, trebuie afișat `1`.

O primă soluție completă ar fi:

```
cin>>n;
if (n == 0)
    cout<<1;
else {
    s = 0;
    while(n != 0){
        s = s + 1;
        n = n/10;
    }
    cout<<s;
}
```

O altă soluție, mai compactă, ar fi:

```
cin>>n;
```

```
if (n == 0) {
    cout<<1;
    return 0;
}
s = 0;
while(n != 0){
    s = s + 1;
    n = n/10;
}
cout<<s;
```

În ultimul caz nu mai este necesară indentarea codului la dreapta odată cu folosirea lui return, dar acest mod de scriere poate fi folosit doar dacă dorim ieșirea imediată din program. O altă abordare, pe care o lăsăm ca exercițiu este folosirea unei instrucțiuni do while.

2. **Suma cifrelor.** Citim o valoare naturală, într-o variabilă de tip int. Determinați suma cifrelor ale sale.

```
cin>>n;
s = 0;
while(n != 0){
    s = s + n%10;
    n = n/10;
}
cout<<s;
```

Deci, operația cu cifra curentă este adunarea ei la sumă.

3. **Cifra maximă.** Citim o valoare naturală, într-o variabilă de tip int. Determinați valoarea celei mai mari cifre ale sale.

```
cin>>n;
maxim = 0;
while(n != 0){
    if (n%10 > maxim)
        maxim = n%10;
    n = n/10;
}
cout<<maxim;
```

4. **Cifra minimă.** Citim o valoare naturală, într-o variabilă de tip int. Determinați valoarea celei mai mici cifre ale sale.

```
cin>>n;
minim = 10;
while(n != 0){
    if (n%10 < minim)
        minim = n%10;
    n = n/10;
}
cout<<minim;
```

Observăm că în acest caz inițializarea s-a făcut cu o valoare mai mare. Codul nu oferă bine rezultatul dacă numărul introdus este 0 (caz în care nu se intră în while și variabila minim rămâte cu valoarea 10). Acest caz se tratează separat așa cum s-a arătat mai sus.

Cele patru aplicații de mai sus reprezintă exemplificarea pe cifrele unui număr a unor cerințe clasice de parcurgere a termenilor unui șir, lucru făcut și la capitolul cu structuri repetitive când șirul era alcătuit din valori introduse de la tastatură (numărare, sumă, minim, maxim).

Următorul set de aplicații se referă la construirea altor numere folosind cifre ale unui număr dat.

5. **Oglinditul.** Citim o valoare naturală, într-o variabilă de tip int. Determinați valoarea obținută considerând cifrele în ordine inversă.

Ne bazăm pe următoarea observație: dacă r este o variabilă ce memorează o valoare naturală (de oricâte cifre) iar c este o variabilă ce memorează un număr de o cifră, construcția r = r * 10 + c face ca valoarea r să se modifice alipindu-i-se cifra c. De exemplu, dacă r este 276 și c este 4, atunci r devine 2764. Dacă r este 0 și c este 2, r devine 2. Așadar, parcurgând cifrele în ordinea clasică de mai sus (de la dreapta) și alipindu-le unei variabile care inițial este egală cu 0, obținem oglinditul (răsturnatul) numărului dat.

```
cin>>n;
r = 0;
while (n!=0) {
    r = r * 10 + n%10;
    n = n/10;
}
cout<<r;
```

Trebuie observat că algoritmul anterior nu obține neapărat un rezultat cu același număr de cifre ca numărul dat (pentru cazul în care numărul dat se termină în cifre 0, acestea nu rămân în partea din față a valorii construite).

6. **Construirea de valori cu cifrele în aceeași ordine ca în numărul dat.**

Presupunem că r memorează o valoare naturală iar c o altă valoare dar de o cifră. Ce expresie trebuie scrisă pentru a plasa cifra c în fața cifrelor lui r (si să obținem în r noua valoare) ? Fie $p = 10^k$, unde k reprezintă numărul de cifre ale lui r. Atunci expresia care ne trebuie este: r = c * p + r;

Exemplu: r = 231 (deci si p = 1000) iar c = 7

```
c*p        7000 +
r           231
rezultat   7231
```

Valoarea variabilei p o vom construi pe măsură ce parcurgem cifrele numărului: pornim cu p de la 1 și pe măsură ce eliminăm o cifră din n (și o adăugăm la r) înmulțim un 10 la p.

```
cin>>n;
r = 0;
p = 1;
while (n!=0) {
    r = n%10 * p + r;
    p = p * 10;
```

```
    n = n/10;
}
cout<<r;
```

Codul anterior nu face decât să reconstruiască în `r` valoarea lui `n`. Adică se afișează ceea ce se citește. Pare inutil, nu? Modul de construire este util însă în probleme ca următoarea: Să se construiască valoarea ce se poate obține cu cifrele pare ale lui `n` în ordinea în care apar.

```
cin>>n;
r = 0;
p = 1;
while (n!=0) {
    if (n%2 == 0){
        r = n%10 * p + r;
        p = p * 10;
    }
    n = n/10;
}
cout<<r;
```

Observăm că la apariția unei cifre pare facem ceea ce am descrie mai sus. Este important ca instrucțiunea `p=p*10` să se afle acum în if, adăugarea unui zero la p făcându-se doar când punem o cifră la `r` (analizați, ca exercițiu, ce se întîmplă dacă am pune în afara lui if atribuirea `p=p*10`).

O altă idee este să construim oglinditul folosind doar cifrele pare și apoi valoarea construită să o oglindim. Acest lucru nu fucnționează însă dacă ultimele cifre ale numărului original sunt `0`.

Probleme diverse rezolvate

7. Să se scrie un program care să determine produsul cifrelor impare ale unui număr natural citit de la tastatură. Dacă numărul nu are cifre impare se va afișa `-1`. (#65, pbinfo.ro).

```
#include <iostream>
using namespace std;
int n, p, impare;
int main () {
    cin>>n;
    p = 1;
    impare = 0;
    while (n!=0) {
        if (n%2 == 1) {
            impare++;
            p = p * (n%10);
        }
        n = n/10;
    }
    if (impare != 0)
        cout<<p;
    else
        cout<<-1;
```

```
}
```

Observațiile relevante la codul de mai sus sunt:

- La instrucțiunea `p = p * (n%10)` lipsa parantezelor duce la un rezultat eronat întrucât operatorul `%` se aplică ultimul (fiind de aceeași prioritate cu `*` și aflându-se mai la dreapta) și rezultatul ar avea mereu o singură cifră.
- Dacă am decide să afișăm `-1` dacă `p` rămâne la final `1`, rezultatul ar fi incorect în cazul prezenței doar a cifrei impare `1`. De aceea am ales varianta de a număra separat cifrele impare.

8. **Produsul primelor două cifre.** Se citește un număr n despre care știm că are cel puțin două cifre. Să se afișeze produsul primelor două cifre ale sale. (#2660, pbinfo.ro).

```
#include <iostream>
using namespace std;
long long n;
int main () {
    cin>>n;
    while (n >= 100)
        n/=10;
    cout<<n/10 * (n%10);
    return 0;
}
```

În prima etapă eliminăm cifre din coadă una câte una până când numărul rămâne de două cifre. Cu această ocazie obținem răspunsul și la problema ridicată la începutul materialului, aceea de a calcula prima cifră a numărului (eliminăm cifre cât timp numărul are cel puțin două). O altă soluție a problemei noastre este de a citi direct doar primele două cifre în variabile de tip char.

9. Se citesc `n` numere. Să se calculeze suma ai cărei termeni sunt **cifrele de pe prima poziție** de la fiecare număr citit (#127, pbinfo.ro).

```
#include <iostream>
using namespace std;
int n, x, i, s;
int main () {
    cin>>n;
    for (i=1;i<=n;i++) {
        cin>>x;
        while (x >= 10) {
            x = x/10;
        }
        s = s + x;
    }
    cout<<s;
}
```

Aici algoritmul de parcurgere de cifre (cu scopul de a o obține pe prima) se aplică la fiecare număr din șirul citit. Deci avem două repetiții, una în alta: cea care ne permite preluarea numerelor unul câte unul și cea de parcurgere a cifrelor.

10. **Numere palindrom.** Se citesc n numere. Determinați palindromul maxim și de câte ori apare. Numerele palindrom sunt cele care au aceeași valoare dacă sunt citite de la dreapta (#280, pbinfo.ro).

```
#include <iostream>
using namespace std;
int maxim, x, ap, y, r;
int main () {
    int exista = 0;
    for (;;) {
        cin>>x;
        if (x == 0)
            break;
        r = 0;
        y = x;
        while (y!=0) {
            r = r * 10 + y%10;
            y /= 10;
        }
        if (r == x) {
            exista = 1;
            if (x > maxim) {
                maxim = x;
                ap = 1;
            } else
                if (x == maxim)
                    ap++;
        }
    }
    if (exista!=0)
        cout<<maxim<<" "<<ap;
    else
        cout<<"NU EXISTA";
}
```

Pentru a verifica dacă un număr este palindrom îi construim oglinditul și testăm dacă acesta este egal cu valoarea originală. Înainte de calcul valoarea originală se salvează în altă variabilă pentru a nu o altera. Atenție la inițializările care trebuie făcute în interiorul lui for (pentru a relua calculele pentru fiecare număr).

11. **Cifra de control**. Pentru un număr dat se calculează suma cifrelor, apoi suma cifrelor valorii obținute și tot așa până când se obține un număr de o cifră. Dată fiind o valoare naturală nenulă, se cere calcularea cifrei sale de control (#340, pbinfo.ro).

```
#include <iostream>
using namespace std;
int n, s;
int main() {
    cin>>n;
    while (n>9) {
        s = 0;
        while(n!=0) {
            s += (n%10);
            n /= 10;
```

```
        }
        n = s;
    }
    cout<<n;
    return 0;
}
```

O altă soluție se bazează pe următoarea observație: dacă scriem sirul numerelor naturale și în paralel șirul cifrelor de control pentru fiecare valoare, observăm (și se demonstrează) că șirul cifrelor de control este periodic (cu perioada 9).

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 …
1 2 3 4 5 6 7 8 9  1  2  3  4  5  6  7  8  9  1  2  3 …
```

Așadar, dacă numărul dat este multiplu de 9, rezultatul este 9 iar în caz contrar rezultatul este egal cu restul împărțirii la 9 al valorii date.

```
#include <iostream>
using namespace std;
int n;
int main() {
    cin>>n;
    if (n%9 == 0)
        cout<<9;
    else
        cout<<n%9;
    return 0;
}
```

Observăm că această soluție nu conține repetiții. Este utilă pentru obținerea unui timp de executare mai bun când se cere calcularea cifrei de control pentru fiecare dintre mai multe numere date (repetiția de la cealaltă soluție ar putea conta fiind vorba de mai multe numere).

12. Dat fiind un număr, se construiesc alte două numere, unul cu cifrele din prima jumătate a sa, în ordinea în care apar, celălalt cu cifrele din a doua jumătate. Dacă valoarea dată are număr impar de cifre, cea din mijloc se ignoră. Să se determine valoarea absolută a diferenței celor două numere construite (#442, pbinfo.ro).

```
#include <iostream>
using namespace std;
int n, m, a, b, c, p, i;
int main () {
    cin>>n;
    m = n;
    c = 0;
    while (m!=0) {
        c++;
        m /= 10;
    }
    p = 1;
    for (i=1;i<=c/2;i++)
```

```
        p = p*10;
    if (c % 2 == 0) {
        a = n%p;
        b = n/p;
    } else {
        a = n%p;
        b = n/(p*10);
    }
    if (a > b)
        cout<<a-b;
    else
        cout<<b-a;
    return 0;
}
```

Calculăm numărul de cifre pentru valoarea dată, notat cu `c`. Apoi, calculăm valoarea $p = 10^{c/2}$, unde `c/2` reprezintă partea întreagă de la împărțirea lui `c` la `2`. Din cele discutate la începutul materialului, una dintre valori (cea cu ultimele cifre) se obține cu expresia `n%p`, iar cealaltă cu expresia `n/p` (dacă `n` are număr par de cifre) și cu expresia `n/(p*10)` dacă `n` are număr impar de cifre.