**Interschimbarea valorilor a două variabile**

Se face cu ajutorul unei a treia variabile. De regulă o numim "aux" (de la "auxiliară"):

**aux = a;** Dacă le scriem în linie poate sunt mai ușor de reținut:

**a = b; aux = a; a = b; b = aux;**

**b = aux;** la fel de bine e: **aux = b; b = a; a = aux;**

**Interschimbarea fără o a treia variabilă (aux)**

Se poate aplica DOAR valorilor aritmetice (sau, în mod extins, celor care suportă operații algebrice de adunare și scădere) astfel: (fie **a** și **b** cele două variabile)

Iată două posibile scrieri ale acestora:

**a = a-b; SAU a = a+b;**

**b = a+b; b = a-b;**

**a = b-a; a = a-b;**

**Se poate realiza și folosind operatorii pe biți, mai precis ^ = XOR (sau exclusiv)**

a = (a^b);

b = (a^b);

a = (a^b);

(operația XOR = ^ are următorul tabel de adevăr:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | a XOR b |  | Iată un exemplu de interschimbare cu a=10110110 și b=01101100  a= 10110110 noul a=11011010 a=11011010  b= 01101100 b=01101100 noul b=10110110 (de fapt fostul a)  a^b======== a^b========= a^b========  a=11011010 b=10110110 a=01101100  deci b a luat fostul a deci a a luat fostul b |
| 0 | 0 | 0 |  |
| 0 | 1 | 1 |  |
| 1 | 0 | 1 |  |
| 1 | 1 | 0 |  |

**Algoritmi care operează asupra cifrelor unui număr**

***1. Separarea cifrelor unui număr***

Algoritmul se bazează pe împărțiri succesive la 10.

Practic, luând un număr natural n, prin expresia n%10 obținem ultima sa cifră iar, prin atribuirea n = n/10; se "șterge" ultima lui cifră.

Bazându-ne pe aceste două operații, putem deduce următorul algoritm:

(fie n=numărul căruia îi separăm cifrele)

**do !!** insist pe do..while din cauză că numărul poate fi și 0!!

**{**

**c=n%10;**

**...prelucrăm cifra c...**

**n/=10;**

**}while(n);**

Obs:

1. Am folosit atribuirea "prescurtată" n/=10; care e echivalentă cu n=n/10;

(există astfel de forme prescurtate pentru toate operațiile.

De ex: s+=x; e echiv. cu s=s+x;)

2. Am folosit condiția prescurtată while(n). În C++, orice condiție, dacă OMITE comparatorii, trebuie să subînțelegem (adică adăugăm "în minte") un "!=0". Deci, în cazul de față subînțelegem while(n!=0)

3. În urma aplicării algoritmului, valoarea lui n se distruge. Prin urmare, dacă mai avem nevoie de ea, îi facem o copie înainte de a aplica algoritmul.

4. Cifrele numărului se obțin în ordine inversă (de la dreapta la stânga)

**Tipul de date int**

ocupă 4 octeți (32 biți) și permite memorarea unei valori cuprinse între -231­..231-1, adică

-2147483648.. 2147483647 (deci aprox. ±2 miliarde, deci putem să ne bazăm pe maxim 9 cifre). Depășirea acestui domeniu de valori produce obținerea de valori eronate.

***2. Formarea unui număr când se dau cifrele sale de la stânga la dreapta***

Formarea se face progresiv, din aproape în aproape, mergând pe ideea că, la fiecare pas, la un număr existent se va lipi o nouă cifră.

Fie nr = variabila în care formăm numărul nou.

Avem două etape:

1. Inițializare (trebuie făcută o singură dată, la început):

nr = 0;

2. Lipirea unei cifre "c" la numărul deja existent:

nr = nr\*10 + c;

***3. Formarea unui număr când se dau cifrele sale de la dreapta la stânga***

Formarea se face de asemenea progresiv. Pentru alipirea unei cifre la stânga e necesar ca, pe măsură ce formăm, să mai menținem într-o altă variabilă p o putere a lui 10 convenabilă adăugării unei cifre în față.

Avem două etape:

1. Inițializare ((trebuie făcută o singură dată, la început):

nr = 0;

p = 1;

2. Lipirea unei cifre "c" la numărul deja existent:

nr = nr + c\*p;

p \*= 10;

**Indicații generale pentru problemele cu numere**

În general ne putem descurca cu algoritmii de mai sus pe o serie de algoritmi.

Lucrurile se pot simplifica de multe ori și prin inventarea de diferite artificii. Acestea, de regulă, folosesc ca principiu aritmetic împărțirile și resturile la puteri ale lui 10.

Ex:

**1634**376 / 1000 → 1634 (taie 3 cifre de la coadă)

1634**376** % 1000 → 376 (păstrează doar ultimele 3 cifre)

**De exemplu, pentru a obține prima cifră a unui număr în variabila pc**

pc=număr;

while(pc>=10)

pc/=10;

**Pentru a obține numărul fără prima sa cifră:**

p=1;

while(p\*10<=numar)

p\*=10;

raspuns=numar%p;

**Iată și un exemplu de algoritm pentru a afla cifrele numărului de la stânga la dreapta:**

Fie "numar" = numărul căruia vrem să-i afișăm cifrele.

Ex: numar=52981 vom afla cifrele în ordinea 5 2 9 8 1

p=1;

while(p\*10<=numar)

p\*=10;

while(p)

{

cifra=numar/p%10;

p/=10;

}

!! Ca regulă generală - să vă feriți de dubla oglindire: dacă numărul inițial se termină într‑un zero sau un grup de zerouri, acestea se pierd!!

**Afișarea numerelor reale**

În cazul numerelor reale, afișarea standard din C++ este setată să lucreze cu 6 cifre.

Depășirea acestui prag produce fie rotunjiri la ultimele zecimale afișate (aproximare), fie afișarea în formă științifică.

Ex:

**cout<<12.34; //12.34**

**cout<<12.34567; //12.3457**

**cout<<12.34562; //12.3456**

**cout<<12345678.0<<"\n";//afișează 1.23457e+07**

Dacă vrem să NU mai folosească forma cu "e" (științifică) folosim biblioteca "**<iomanip>**" și funcțiile "**fixed**" și "**seprecision(nr\_zecimale)**" pt. controlarea afișării.

Mai există exerciții la care se cere afișarea fără rotunjire la ultima zecimală (de exemplu să se afișeze cu 2 zecimale, fără rotunjire la ultima).

Știm că setprecision(nr\_zecimale) face rotunjire (nu există funcție care să scape de asta). În acest sens, "ne"rotunjirea ultimei zecimale se face aritmetic.

Pe exemplul cu 2 zecimale, dacă avem de afișat o valoare reală **x** fără rotunjire la a doua zecimală:

**cout<<fixed<<setprecision(2)<<(int)(x\*100)/100.0;**