

Pavel Florin Moraru

INFORMATICA
pentru
LICEU
și
BACALAUREAT

Clasele a IX-a și a X-a, neintensiv

profil Matematică – Informatică,
varianta C++

Colecția
Donaris – Info



Copyright © 2003 – Editura Donaris, Sibiu
Toate drepturile sunt rezervate autorilor și Editurii Donaris.
Nici o parte a acestei lucrări nu poate fi copiată sau reprodusă
fără permisiunea scrisă a Editurii Donaris.

Culegere, tehnoredactare computerizată și design:

• Laurențiu Codău • Magda Codău

Coperta colecției "Donaris – Info":

• Attila Biblia

DESCRIEREA CIP A BIBLIOTECII
NAȚIONALE A ROMÂNIEI

Moraru, Pavel-Florin
Informatica pentru liceu și Bacalaureat
Clasele a IX-a și a X-a, varianta C++, neintensiv (nr. 14)
ISBN: 978 - 606 - 8014 - 32-6, 288 pagini



EDITURA DONARIS

Tel: 0752 / 79.47.65

E-mail: editura_donaris@yahoo.com

ISBN: 978-606-8014-32-6

Dedic această carte părinților mei,
în semn de recunoștință
pentru tot ceea ce au făcut pentru mine:

- ▲ Post-mortem, tatălui meu, **ING. PAVEL MORARU**
- ▲ Mamei mele, PROF. EUGENIA MORARU

Autorul



Capitolul I: Algoritmi



I.1. Generalități despre algoritmi



I.1.1. Noțiunea de algoritm

Așa cum deja știți, pentru a rezolva o problemă cu ajutorul calculatorului trebuie să scriem un program într-un anumit limbaj de programare. Dar înainte de a ajunge la program, trebuie să proiectăm rezolvarea problemei.

Definiție:

Succesiunea etapelor rezolvării unei probleme, în ordinea parcurgerii lor, prin care se prelucrează un set de date de intrare, în scopul obținerii unor date de ieșire (rezultate), se numește **algoritmul problemei**.

Etapile rezolvării se mai numesc și **pașii algoritmului**.

Orice informație, de orice gen, care descrie un anumit fenomen, eveniment etc., poate fi codificată sub forma unei date. Așa cum reiese și din definiția de mai sus, în cadrul unui algoritm întâlnim două tipuri de date:

- ☐ **Datele de intrare** trebuie să fie cunoscute la "intrarea în algoritm", adică la începutul execuției algoritmului.
- ☐ **Datele de ieșire** sunt de fapt rezultatele problemei pe care o rezolvă algoritmul respectiv, rezultate obținute în timpul execuției algoritmului, prin prelucrarea datelor de intrare.

Orice algoritm trebuie să îndeplinească câteva cerințe fundamentale:

- a) să fie cât mai general, adică să acopere toate situațiile care pot interveni în rezolvarea problemei respective;
- b) să fie bine-definit. Altfel spus, cerințele problemei să fie formulate clar, fără ambiguități, iar datele de care dispune algoritmul să fie suficiente pentru rezolvarea problemei;
- c) să fie finit. Algoritmul unei probleme trebuie proiectat de așa manieră încât să existe certitudinea că la un moment dat se va încheia cu succes execuția lui, obținându-se toate soluțiile problemei;
- d) să fie rezolvabil. În traducere, orice algoritm trebuie să aibă cel puțin o soluție.



I.1.2. Obiectele cu care lucrează algoritmi

Orice algoritm folosește anumite "obiecte" asupra cărora sunt permise anumite operații. Aceste obiecte sunt de două tipuri: **date** și **expresii**.

Exemplu:

Să încercăm să ne imaginăm un algoritm pentru traversarea unei străzi într-o intersecție semaforizată. Prima acțiune pe care o executăm în cadrul algoritmului este aceea de a ne uita la culoarea semaforului. Creierul uman înregistrează culoarea semaforului ca pe o informație. Prin prelucrarea acestei informații, hotărâm ce acțiune executăm în continuare: dacă semaforul arată "verde", atunci ne angajăm în traversarea străzii, în caz contrar... mai așteptăm. Informația numită "culoarea semaforului" este un obiect cu care lucrează algoritmul pentru traversarea străzii. Așa cum am spus mai înainte, orice informație se poate codifica (reprezenta) sub forma unei **date**. Astfel, în exemplul nostru, informația numită *culoarea semaforului* poate fi reprezentată sub forma unui șir de caractere cuprins între ghilimele: "rosu" respectiv "verde". De ce este necesară această codificare? Dacă ar fi să elaborăm un program pentru algoritmul de traversare a străzii, trebuie să ne gândim că un calculator nu poate percepe culorile așa cum o face creierul uman! Calculatorul trebuie să memoreze informația numită culoare sub forma unei date, pe care să o poată prelucra în maniera în care știe el să gândească.

Datele unui algoritm pot fi de două tipuri: **constante** și **variabile**.

Constante

Definiție:

O constantă este o dată a cărei valoare nu se modifică în timpul execuției algoritmului.

Constantele sunt de mai multe tipuri:

- a) **Constantele întregi.** Sunt numere din mulțimea $\mathbb{Z}$, adică numere întregi cu sau fără semn. *Exemple:* 232, +87, -1405.
- b) **Constante reale.** Sunt valori raționale, cu virgulă zecimală, dar care au un număr finit de zecimale. Mai notăm că *virgula zecimală se reprezintă prin caracterul "punct"*. *Exemple:* 2.34, -45.26
- c) **Constante caracter.** Sunt caractere cuprinse între apostrofuri. Acestea pot fi litere mari și mici, caractere-cifra și caractere speciale. *Exemple:* 'A', 'm', '\$', '9' etc.
- d) **Constantele șir de caractere.** Sunt șiruri de caractere cuprinse între ghilimele, care pot conține orice caractere, inclusiv caractere speciale și caracterul "spațiu". *Exemple:* "program", "Borland C++ 7.0", "AB#7", etc.

În programe vom folosi foarte mult așa-numitele "**constante întregi cu sens logic (boolean)**". Ce rol au acestea? În cadrul unui algoritm putem întâlni

condiții, care pot fi îndeplinite sau nu, și în funcție de care se vor decide acțiunile următoare. Orice condiție se caracterizează printr-o așa numită "*valoare de adevăr*", care ne spune dacă respectiva condiție este îndeplinită sau nu. Astfel, o condiție poate avea valoarea de adevăr "**ADEVĂRAT**" (condiție îndeplinită, adevărată), sau "**FALS**" (condiție neîndeplinită, falsă). Deoarece în limbajul C++ nu există un tip de date care să desemneze valorile logice, vom simula cele două valori de adevăr posibile ale unei condiții prin numerele întregi 1 respectiv 0. Mai exact, numărul întreg 1 corespunde valorii de adevăr "**ADEVĂRAT**", iar numărul întreg 0 identifică valoarea de adevăr "**FALS**". Mai multe detalii pe această temă veți învăța în capitolele următoare.

Variabile

Definiție:

Numim **variabilă**, un obiect caracterizat prin trei atribute, tip, adresă și valoare, cu proprietatea că atributul valoare este modificabil.

Exemplu:

Fie funcția matematică $f: [-10, 10] \rightarrow \mathbb{R}$, $f(x) = 2x + 1$.

Aceasta este definită prin: *domeniu de definiție* $[-10, 10]$ (mulțimea valorilor pe care le poate lua x), *codomeniul* $\mathbb{R}$ (mulțimea valorilor funcției $f(x)$) și *legea de asociere* $f(x) = 2x + 1$. Despre x putem spune că poate lua orice valoare din domeniu, și fiecărei valori a lui x îi corespunde o valoare a funcției. De exemplu, pentru $x = -3$ obținem $f(-3) = 2 \cdot (-3) + 1 = -5$, pentru $x = 2$ avem $f(2) = 5$ etc. Astfel, x este o *variabilă* pentru că valoarea sa se poate modifica în orice moment. Mai exact, dacă am proiecta un algoritm pentru calculul valorilor funcției de mai sus, atunci valoarea lui x s-ar modifica în cursul execuției algoritmului. După cum observați, sensul noțiunii de variabilă este cel cunoscut din matematică.

Tipul unei variabile stabilește mulțimea valorilor pe care le poate primi variabila respectivă. La nivel de algoritmi vom folosi patru tipuri, cele pe care le cunoașteți deja din lecția "Constante". Le sintetizăm în continuare, dar fără a insista prea mult asupra lor, deoarece le vom relua în capitolul III, în care vom prezenta elementele de bază ale limbajului C++.

Denumirea tipului	Cum pot fi valorile variabilelor de tipul respectiv
Tipul întreg	Numere întregi cu sau fără semn
Tipul real	Numere reale (cu virgulă zecimală reprezentată prin caracterul "punct")
Tipul caracter	Caractere cuprinse între apostrofuri
Tipul șir de caractere	Șiruri de caractere cuprinse între apostrofuri

La fel ca și în matematică, orice variabilă trebuie să aibă un nume, căruia îi vom spune de acum încolo identificator de variabilă. Evident în cadrul unui algoritm nu pot exista două variabile cu același identificator.

Expresii

În scopul efectuării calculelor, algoritmi folosesc **expresii**. O expresie este alcătuită dintr-unul sau mai mulți *operandi* legați între ei prin *operatori*.

Operandii reprezintă valorile care intră în calcul. Ei pot fi *constante* și *variabile*. **Operatorii** desemnează operațiile care se execută în cadrul expresiei.

Vom vedea mai târziu că într-un program C++ unii dintre operatori se simbolizează altfel decât îi cunoaștem din matematică. Desigur că atunci când scriem un algoritm pe hârtie putem nota operatorii cum vrem. Chiar dacă deocamdată ne ocupăm numai de algoritmi, vom introduce de la bun început niște notații apropiate de sintaxa limbajului C++, pentru a evita riscul unor confuzii ulterioare.

În timpul execuției algoritmului expresiile sunt evaluate. Evaluarea unei expresii are loc astfel: *se înlocuiesc variabilele cu valorile lor și se efectuează calculele, obținându-se astfel valoarea expresiei*.

Expresiile se împart în două categorii: *expresii aritmetice* și *expresii logice*. Principalele categorii de operatori sunt: *operatori aritmetici*, *logici* și *relaționali*.

Expresii aritmetice

Sunt cele care efectuează operații aritmetice având ca rezultat un număr. În continuare vom nota operatorii aritmetici așa cum sunt ei definiți în limbajul C++. Mai mult, vom încerca să vă obișnuim de pe acum să respectați câteva reguli pe care le impune acest limbaj în ceea ce privește folosirea operatorilor.

Operatorii aritmetici sunt:

+ (adunare) * (înmulțire) - (scădere) / (împărțire)

Exemplu:

- Expresia $\frac{a+b}{2}$ din matematică se scrie într-un program sub forma $(a+b)/2$.

Să vedem cum procedează calculatorul pentru a evalua expresia în cazul în care variabilele au valorile $a=5$ și $b=7$: așa cum am spus, se înlocuiesc variabilele cu valorile lor și se efectuează calculele, obținându-se valoarea expresiei; deci valoarea expresiei $(a+b)/2$ este $(5+7)/2$, adică 6. Operatorul "/" a înlocuit linia de fracție.

Parantezele sunt absolut necesare. Există o așa numită prioritate a operatorilor, ce stabilește ordinea în care se execută operațiile în timpul evaluării unei expresii. Conform acesteia, se execută mai întâi împărțirile și înmulțirile, apoi adunările și scăderile. Așadar, în absența parantezelor, expresia $a+b/2$ este echivalentă cu "formula" $a + \frac{b}{2}$.

Expresii logice (booleene)

Așa cum am mai spus, în algoritmi întâlnim deseori *condiții*. O condiție poate fi scrisă sub forma unei expresii logice. Valoarea unei expresii logice

reprezintă valoarea de adevăr a condiției aferente, și poate fi **"ADEVĂRAT"** (condiție îndeplinită, adevărată) și **"FALS"** (condiție neîndeplinită, falsă). Am arătat în lecția "Constante" că în limbajul C++ vom simula cele două valori de adevăr posibile ale unei condiții prin numerele întregi 1 respectiv 0. Astfel, o expresie logică poate avea valoarea:

- 1, în cazul în care condiția pe care o definește este adevărată;
- 0, în situația în care condiția pe care o descrie este falsă.

O expresie logică este construită de obicei cu ajutorul operatorilor relaționali. Am precizat deja faptul că în limbajul C++ unii operatori sunt simbolizați diferit față de notația cunoscută din matematică. Pentru a vă obișnui de la bun început cu sintaxa limbajului C++, vom defini încă de pe acum operatorii așa cum sunt ei implementați în limbaj. Operatorii relaționali ai limbajului C++ sunt: "==" (operatorul de testare al egalității), "<" ("mai mic"), ">" ("mai mare"), "<=" ("mai mic sau egal"), ">=" ("mai mare sau egal"), "!=" ("diferit").

Exemplu:

- Dacă a și b sunt variabile numerice (întregi sau reale), $a-10>=b$ este o expresie logică: pentru $a=12$ și $b=5$, expresia are valoarea 0, corespunzătoare valorii de adevăr **FALS** (" $12-10>=5$? nu"); în schimb, pentru $a=8$ și $b=6$, expresia are valoarea 1, corespunzătoare valorii de adevăr **ADEVĂRAT** (" $8-10>=6$? nu").

Mai multe expresii logice pot fi "combinate" cu ajutorul *operatorilor logici*, obținându-se o așa numită *"expresie logică compusă"*. Expresiile logice "elementare" ce alcătuiesc expresia compusă pot fi scrise între paranteze. În mod implicit, operațiile ce apar în cadrul unei expresii, se execută într-o anumită ordine, dată de așa-numita "prioritate a operatorilor". Despre acest subiect am pomenit deja în cadrul exemplului dat în paragraful "Expresii aritmetice", precizând atunci faptul că rolul parantezelor într-o expresie este tocmai acela de a schimba ordinea implicită în care se execută operațiile. Operatorii logici pe care-i vom folosi în acest capitol sunt **ȘI**, **SAU** (pentru aceștia nu vom introduce încă notația lor din limbajul C++, preferând deocamdată aceste două cuvinte din limba română care sunt mult mai sugestive).

Exemplu:

Fie a și b două variabile de tip întreg. Condiția ca "numerele a și b să fie ambele pozitive", poate fi scrisă astfel:

- $a>0$ ȘI $b>0$
- $(a>0)$ ȘI $(b>0)$

În a doua formă, expresiile elementare ce compun expresia logică compusă, adică " $a>0$ " și " $b>0$ " au fost cuprinse între paranteze rotunde. În cazul de față, parantezele nu sunt necesare, deoarece în mod implicit compilatorul limbajului C++ efectuează operațiile în ordinea în care dorim: mai întâi evaluează expresiile elementare " $a>0$ " respectiv " $b>0$ ", apoi aplică "**ȘI** logic" între valorile acestora.

Pentru ca expresia " $a>0$ ȘI $b>0$ " să fie adevărată, este necesar ca expresiile componente " $a>0$ ", " $b>0$ " să fie adevărate în același timp.

- pentru $a=3, b=8$, expresia are valoarea 1 (cu sens de "ADEVĂRAT"); (" $3>0$ și $8>0$? da");
 - pentru $a=4, b=-2$, expresia are valoarea 0 (cu sens de "FALS"); (nu este " $b>0$ ").
 - $a>0$ SAU $b>0 \rightarrow$ descrie condiția ca cel puțin unul dintre numerele a, b să fie pozitive.
- Pentru ca expresia " $a>0$ SAU $b>0$ " să aibă valoarea 1 ("ADEVĂRAT"), trebuie ca cel puțin una dintre expresiile " $a>0$ ", " $b>0$ " să fie adevărată.
- pentru $a=4, b=-2$ expresia are valoarea 1 ("ADEVĂRAT");
 - pentru $a=-3, b=-5$ expresia are valoarea 0 ("FALS").



I.2. Reprezentarea algoritmilor. Pseudocodul.

Așa cum am spus deja, algoritmul unei probleme este alcătuit dintr-o succesiune de pași ce reprezintă etapele rezolvării problemei. În cadrul fiecărui pas se execută una sau mai multe acțiuni.

Scopul elaborării algoritmului unei probleme este acela de a scrie un program într-un anumit limbaj de programare. Dar dacă avem de-a face cu o problemă mai complexă, înainte de a scrie programul este bine să schițăm pașii algoritmului. În acest scop avem la dispoziție o formă foarte accesibilă de reprezentare a algoritmilor, și anume **pseudocodul**.

Un **limbaj pseudocod** se prezintă sub formă de text și se bazează pe niște așa-numite *cuvinte cheie*. Fiecare cuvânt cheie identifică în mod unic un anumit tip de acțiune.

Acțiunile algoritmului se reprezintă în pseudocod prin ceea ce numim **instrucțiuni**.

Ansamblul cuvintelor cheie împreună cu regulile care trebuie respectate în folosirea lor, alcătuiește ceea ce numim **sintaxa limbajului-pseudocod**.

Există o mare diversitate de limbaje-pseudocod. Practic, fiecare programator își poate proiecta propriul pseudocod, definind cuvintele cheie ale acestuia și impunând niște reguli de sintaxă.

În cele ce urmează vom folosi o variantă de pseudocod în care cuvintele cheie provin din limba română, iar regulile de sintaxă sunt cele pe care le impune limbajul C++. Deocamdată trebuie să reținem două dintre aceste reguli:

- asemeni limbajului C++, în pseudocod nu vom face deosebire între litere mari și litere mici;
- comentariile, prin care descriem pe scurt acțiunile algoritmului (instrucțiunile pseudocodului), vor fi precedate de caracterele "/*".



I.2.1. Primul exemplu de pseudocod

Vom începe cu un exemplu foarte simplu: algoritmul pentru calculul ariei unui triunghi cu formula $A = \frac{b \cdot h}{2}$, unde b este o latură a triunghiului (baza), iar h este înălțimea corespunzătoare.

Să vedem ce instrucțiuni ar trebui să execute programul scris după acest algoritm, și cum se reprezintă ele în pseudocod. Le vom enumera în ordinea "derulării" lor.

Mai întâi calculatorul trebuie să preia valorile laturii bazei și înălțimii triunghiului, pe care le va memora în niște variabile simbolizate ca în formulă. Vom spune așadar "baza b și înălțimea h ". Acestea sunt *datele de intrare* ale algoritmului. *De ce se utilizează variabile ?* Un program eficient și general este acela capabil să calculeze aria oricărui triunghi, adică *pentru orice valori ale bazei și înălțimii*. Cu alte cuvinte, vrem ca *de fiecare dată când se execută, programul să poată calcula aria altui triunghi, pentru alte valori ale bazei și înălțimii*. Cum anume facem să se întâmple așa ?

În timpul execuției programului, vom introduce de la tastatură valorile bazei și înălțimii, că valori pe care calculatorul le va prelua și le va memora în variabilele b respectiv h . În sens figurat spus, "calculatorul citește datele de intrare de la tastatură". Această formulare, care pare că dă viață calculatorului, este atât de consacrată în informatică, încât o vom folosi permanent de acum încolo.

În pseudocod, o instrucțiune de citire a datelor de intrare începe cu cuvântul cheie "citește" urmat de numele variabilelor care se citesc separate prin virgulă, și se încheie cu caracterul "punct și virgulă". În exemplul nostru această instrucțiune este:

citește b, h ;

Evident că puteam folosi două instrucțiuni de citire, câte una pentru fiecare variabilă:

citește b ;
citește h ;

De cele mai multe ori este necesar ca datele de intrare să respecte niște *condiții*, pe care calculatorul trebuie să le testeze. În exemplul nostru, după ce a preluat prin citire de la tastatură valorile bazei și înălțimii (pe care le-a memorat în variabilele b și h), calculatorul trebuie să verifice dacă acestea sunt ambele pozitive (nu există triunghi cu bază sau înălțime negativă sau nulă). Altfel spus, condiția testată este "*baza b mai mare ca zero și înălțimea h mai mare ca zero*". Evident, condiția respectivă va fi descrisă printr-o expresie logică. În pseudocod, această instrucțiune se reprezintă sub forma:

dacă (<condiție>)

Și în ceea ce privește această instrucțiune, am impus în pseudocod o regulă de sintaxă a limbajului C++: condiția <condiție> se scrie între paranteze rotunde.

În exemplul nostru vom scrie:

dacă ($b>0$ și $h>0$) atunci



Capitolul II: Principiile programării structurate



II.1. Noțiunea de structură

Reamintim:

În cadrul unui algoritm se pot efectua trei tipuri de operații: de intrare / ieșire, de atribuire (calcul) și de decizie. Algoritmul efectuează aceste operații în scopul transformării datelor de intrare în date de ieșire. Operațiile algoritmului se reprezintă în pseudocod prin instrucțiuni.

Apare însă firesc întrebarea: în ce ordine se execută operațiile în cadrul unui algoritm? În mod normal succesiv, de la prima până la ultima:

operația 1 → operația 2 → ... → operația n

Dar într-un program pot apărea ramificații și "salturi" de la o instrucțiune la alta, care schimbă această ordine. Mai mult, vom vedea în continuare că există și posibilitatea de a repeta de mai multe ori execuția unor acțiuni. Prin urmare, orice algoritm se caracterizează printr-o anumită structură, care indică ordinea execuției operațiilor. Mai exact, **prin structură înțelegem o formă de îmbinare a operațiilor cu care lucrează algoritmi.**

Programarea structurată este de o importanță vitală în realizarea unor programe cât mai clare și mai ușor de urmărit, mai ales în cadrul algoritmilor lungi și complexi.

S-a demonstrat în mod incontestabil că pentru realizarea oricărui program sunt suficiente trei tipuri de structuri:

- ♦ liniară;
- ♦ alternativă (de selecție);
- ♦ repetitivă.

În continuare ne propunem să analizăm pe rând cele trei tipuri de structuri enumerate.



II.2. Structura liniară

În primul exemplu de pseudocod prezentat în capitolul I, cel al algoritmului pentru calculul ariei unui triunghi, am întâlnit toate tipurile de operații pe care le poate efectua un algoritm: citirea datelor de intrare (baza b și înălțimea h a triunghiului), luarea unei decizii în funcție de valoarea de adevăr a condiției " $(b > 0)$ și $(h > 0)$ ", calculul ariei triunghiului (atribuirea $A \leftarrow (b \cdot h) / 2$), afișarea datelor de ieșire (aria A).

Vom defini structura liniară astfel:

- ♦ Orice operație din categoria celor mai sus exemplificate constituie o structură liniară.
- ♦ Dacă s_1 și s_2 sunt structuri de orice tip, atunci este o structură liniară.
- ♦ Generalizând afirmația anterioară, ajungem la structura liniară alăturată.

s_1
 s_2

s_1
 s_2
...
 s_n

Demonstrația este foarte simplă. Dacă s_1 și s_2 sunt structuri liniare, atunci și ansamblul (s_1, s_2) este o structură liniară. Apoi, dacă s_3 este tot o structură liniară, avem un ansamblu de două structuri liniare, (s_1, s_2) și (s_3) . Aceasta, conform afirmației anterioare, alcătuiește tot o structură liniară, (s_1, s_2, s_3) . Analog se demonstrează că (s_1, s_2, s_3, s_4) este tot o structură liniară ș.a.m.d.



II.3. Structura alternativă (de selecție)



II.3.1. Structura de selecție simplă

Structura de selecție simplă, mai numită și structura de tip "dacă-altfel", introduce o ramificație în program, în funcție de valoarea de adevăr a unei condiții. Mai concret, se testează o condiție, formulată cu ajutorul unei expresii logice, pe care o denumim generic prin meta-simbolul **<expresie>**. Această expresie logică poate avea, după cum bine știți deja, doar două valori: 1 (condiție adevărată) sau 0 (condiție falsă). În funcție de valoarea întâlnită în momentul testării, se va executa una din cele două "ramuri" posibile ale structurii de selecție simplă, și anume:

- dacă **<expresie>** are valoarea 1, adică în cazul în care condiția aferentă este adevărată, se execută o anumită secvență de instrucțiuni, pe care o denumim simbolic prin **<secvența_1>**. Această secvență se numește "corpul ramurii "dacă";
- în caz contrar (condiția nu este îndeplinită iar **<expresie>** are valoarea FALSE), se execută o altă secvență de instrucțiuni, pe care o codificăm prin **<secvența_2>** (corpul ramurii "altfel").

În pseudocod, structura de selecție simplă descrisă mai sus ar putea fi reprezentată astfel:

```

dacă (<expresie>) atunci
    <secvența_1>
altfel
    <secvența_2>
    
```

Așa cum am mai spus și în capitolul I, nu există un limbaj pseudocod standard, fiecare utilizator având libertatea de a-și stabili propriul set de reguli de sintaxă. Pentru a vă obișnui de la bun început cu regulile de sintaxă ale limbajului C++, le vom introduce rând pe rând și în pseudocod. Deocamdată stabilim două astfel de "legi":

- la sfârșitul fiecărei instrucțiuni de atribuire, citire și afișare, se pune caracterul "punct și virgulă";
- expresia logică <expresie> din linia "dacă" se cuprinde între două paranteze rotunde.

Exemplu:

Ne propunem să proiectăm pseudocodul unui algoritm care determină și afișează maximul a două numere întregi **a** și **b** citite de la tastatură.

Citim de la tastatură numerele **a** și **b**. Apoi se comparăm numerele, memorând pe cel mai mare dintre ele în variabila **max**.

- dacă **a** este mai mare sau egal cu **b** (condiție care se scrie prin expresia logică "**a** >= **b**") atunci maximul este **a**, adică facem atribuirea **max** ← **a**;
- în caz contrar maximul va fi evident **b**, și marcăm acest fapt prin atribuirea **max** ← **b**.

În final se afișăm maximul **max**.

Observații:

- În cazul în care secvența de instrucțiuni <secvența_1> sau <secvența_2> conține mai mult de o instrucțiune, ea trebuie cuprinsă între cuvintele cheie "început" și "sfârșit". În exemplul anterior, fiecare dintre cele două secvențe era alcătuită doar din câte o instrucțiune de atribuire, "**max** ← **a**", respectiv "**max** ← **b**";
- Pentru ca un pseudocod să arate cât mai limpede, recomandăm ca instrucțiunile ce alcătuiesc corpurile "ramurilor" "dacă" și "altfel" să fie aliniate ceva mai înspre interior. Nu este obligatoriu, dar este mult mai sugestiv!

Într-o structură de selecție simplă de tipul "dacă-altfel", ramura "altfel" poate să lipsească. Asta înseamnă că în cazul în care condiția nu este îndeplinită, pur și simplu nu se execută nimic.

Exemplu:

Iată în continuare o altă variantă a algoritmului pentru determinarea maximului a două numere date **a** și **b**, în care folosim o structură de selecție fără ramura "altfel".

Evident, și de această dată trebuie să începem cu citirea de la tastatură a numerelor **a** și **b**. Presupunem mai întâi că cea mai mare dintre cele două valori este **a**, făcând atribuirea **max** ← **a**. Apoi verificăm posibilitatea contrară, adică testăm dacă nu cumva totuși **b** este mai mare decât **a**; în caz afirmativ, evident maximul celor două numere va fi **b**, și marcăm acest fapt prin atribuirea **max** ← **b**.

În caz de egalitate, maximul poate fi oricare dintre cele două numere, iar în varianta noastră variabila **max** va rămâne cu valoarea inițială **a**.

```

citește a,b;
dacă (a>=b) atunci
    max ← a;
altfel
    max ← b;
scrie max;

```

Aplicație

R.II.1. Aria unui triunghi.

Scrieți pseudocodul unui algoritm care citește de la tastatură două numere reale și strict pozitive **b** și **h**, reprezentând baza, respectiv înălțimea unui triunghi, apoi calculează și afișează aria triunghiului. În cazul în care valorile introduse nu pot reprezenta latura și înălțimea unui triunghi, se va afișa mesaj de eroare. Reamintim formula ariei triunghiului: $A = (b \cdot h) / 2$.

Rezolvare

Orice algoritm trebuie să înceapă cu citirea datelor de intrare, în cazul de față baza triunghiului (adică o latură) și înălțimea corespunzătoare. Așa cum am arătat în lecția introductivă din capitolul I, instrucțiunea necesară este {citește **b**, **h**; }.

Urmează testarea condițiilor pe care trebuie să le îndeplinească datele de intrare. În problema noastră, fiind vorba despre mărimi geometrice într-un triunghi, valorile lui **b** și **h** trebuie să fie strict pozitive. Această testare se realizează cu ajutorul unei instrucțiuni de selecție simplă de tipul "dacă-altfel", care va introduce o ramificație în program. Întrucât baza și înălțimea trebuie să fie ambele mai mari ca 0 în același timp, cele două condiții vor fi "legate" prin operatorul "și logic", iar expresia logică compusă aferentă condiției din linia "dacă" este "**b** > 0 și **h** > 0" (pusă între paranteze, așa cum știți deja). Așadar:

- ✓ Dacă este îndeplinită această condiție atunci:
 - calculăm aria triunghiului; în acest scop, atribuim unei variabile **A** valoarea expresiei $(b \cdot h) / 2$ ce reprezintă aria;
 - afișăm valoarea variabilei **A**, cu ajutorul cuvântului cheie "scrie".
- ✓ În caz contrar, tipărim mesajul "date incorecte".

```

citește b,h;
dacă (b>0 și h>0) atunci
    început
        A ← (b*h) / 2;
        scrie A;
    sfârșit
altfel
    scrie 'Date incorecte';

```

Încercați singuri!

- Tema nr. T.1.

1. Scrieți câte un algoritm care determină și afișează:
 - a) minimul a două numere date.
 - b) maximul a trei numere date.
 - c) minimul a trei numere date.

2. Scrieți câte un algoritm care calculează și afișează:

- a) volumul unui trunchi de con $v = \frac{\pi \cdot h}{3} \cdot (R_1^2 + R_2^2 + R_1 \cdot R_2)$.

Condiții: $R_1, R_2, h > 0$.

- b) Aria unui paralelogram $A = d_1 \cdot d_2 \cdot \sin \alpha$.

Condiții: $d_1, d_2 > 0$; $\alpha \in (0, \frac{\pi}{2})$.

- c) Aria totală a unui cilindru circular drept $A = 2 \cdot \pi \cdot R \cdot (R + G)$.
Condiții: $R, G > 0$.
- d) Aria unui trapez oarecare $A = \frac{(b_1 + b_2) \cdot h}{2}$.
Condiții: $b_1, b_2, h > 0$.

Structuri de selecție imbricate

Într-o structură de selecție simplă, oricare dintre secvențele de instrucțiuni de pe cele două ramuri ale sale poate conține la rândul ei alte structuri de același gen. Se obține astfel o așa-numită *structură selectivă imbricată*. În acest caz, fiecare ramură "altfel" se asociază ultimei ramuri "dacă-atunci" dinaintea sa care nu a fost încă asociată, așa cum arată schema din figura. Dacă dorim să schimbăm această asociere implicită, vom folosi cuvintele cheie "început" și "sfârșit".

```

    dacă (<c1>) atunci
    ┌   dacă (<c2>) atunci
    │   <s1>
    │   altfel
    │   <s2>
    └   altfel
        <s3>

```

Exemple concrete de astfel de structuri imbricate veți întâlni în toate aplicațiile care urmează.

Aplicație

R.II.2. Interpretarea unui pseudocod dat

Se dă secvența de algoritm de mai jos, prezentată în pseudocod, în care toate variabilele identifică numere reale:

citește a, b, c ; dacă $(a > b + c)$ atunci început dacă $(b > c)$
atunci $x \leftarrow (a + b) / (2 \cdot c)$; altfel $x \leftarrow 0$; sfârșit altfel
 $x \leftarrow (a / c + b / c) / 2$; scrie x ;

a) Rescrieți secvența, folosind o așezare în pagină mai "aspectuasă", cu liniile de algoritm scrise una sub alta, și aliniate astfel încât să se vadă fiecare ramură "altfel" a cărei ramuri "dacă" îi corespunde.

b) Deduceți ce se afișează în urma execuției secvenței, dacă valorile citite de la tastatură sunt $a=9, b=5$ și $c=2$.

c) Găsiți un set de valori pentru variabilele a, b și c , astfel încât, în urma execuției algoritmului, să se afișeze valoarea 0.

d) Rescrieți algoritmul într-un mod mai scurt, folosind un număr mai mic de ramuri de tip "dacă-altfel".

e) Dacă din algoritm am șterge cuvintele "început" și "sfârșit", efectul execuției algoritmului s-ar schimba, sau ar rămâne același? Justificați răspunsul.

Rezolvare

a)

```

citește a,b,c;
dacă (a>b+c) atunci
    început
    dacă (b>c) atunci
        x ← (a+b) / (2*c);
    altfel
        x ← 0;
    sfârșit
altfel
    x ← (a/c+b/c) / 2;
scrie x;

```

b)

Pentru $a=9, b=5$ și $c=2$, avem:

– condiția " $a > b + c$ " din prima instrucțiune de selecție este îndeplinită pentru aceste valori ($9 > 5 + 2$), deci se execută ramura "dacă" a acesteia, unde întâlnim o altă instrucțiune de selecție de tipul "dacă-altfel";

– aici condiția " $b > c$ " este la rândul ei adevărată ($5 > 2$), motiv pentru care se execută atribuirea $\{x \leftarrow (a + b) / (2 \cdot c)\}$, de unde rezultă $x = (9 + 5) / (2 \cdot 2) = 14 / 4 = 3.5$. Variabila x nu mai suferă nici o modificare a valorii sale, întrucât celelalte ramuri ale instrucțiunilor "dacă-altfel" imbricate nu sunt atinse. În concluzie, programul va afișa valoarea 3.5.

c)

Singurul loc din secvență în care găsim o atribuire prin care variabila x primește valoarea 0, este pe ramura "dacă" a primei instrucțiuni de selecție și pe ramura "altfel" a celei de-a doua astfel de instrucțiuni. Cu alte cuvinte, valorile alese pentru a, b și c trebuie să îndeplinească prima condiție " $a > b + c$ ", și să nu îndeplinească cea de-a doua condiție " $b > c$ ". Un exemplu de astfel de valori ar fi $a=9, b=3$ și $c=4$, dar, evident, nu singurele.

d)

În secvența dată, avem trei instrucțiuni prin care variabilei x i se atribuie o anumită valoare, dar, dintre acestea, două sunt echivalente. Este vorba despre instrucțiunile $x \leftarrow (a + b) / (2 \cdot c)$ și $x \leftarrow (a / c + b / c) / 2$, cauza fiind faptul că expresiile atribuite, $(a + b) / (2 \cdot c)$, respectiv $(a / c + b / c) / 2$, înseamnă de fapt același lucru. Într-adevăr, nu este greu de observat că expresia $(a / c + b / c) / 2$ se poate scrie și sub forma $(a + b) / c / 2$, adică $(a + b) / (2 \cdot c)$. Prin urmare, cele două ramuri care atribuie variabilei x aceeași valoare pot fi comasate între ele. Care sunt situațiile ce conduc la aceste două atribuiri echivalente?

– una se petrece atunci când se intră pe ramura "dacă" a primei instrucțiuni de selecție și în același timp pe ramura "altfel" a celei de-a doua, adică atunci când " $a > b + c$ și $b > c$ ";

– cealaltă are loc atunci când se intră pe ramura "altfel" a primei instrucțiuni de selecție, adică pentru cazul contrar lui " $a > b + c$ ", care se traduce prin " $a \leq b + c$ ".

Comasarea celor două posibilități într-o singură linie "dacă" se face utilizând operatorul "SAU logic" (poate să se întâmple ori una ori cealaltă dintre alternative, nu amândouă simultan). Așadar:

– dacă $(a > b + c \text{ și } b \geq c)$ SAU $(a \leq b + c)$ atunci lui x i se atribuie valoare expresiei $(a + b) / (2 * c)$;

– în caz contrar, variabila x primește valoarea 0.

Lăsăm ca exercițiu rescrierea secvenței, operațiune care după aceste explicații ar trebui să fie o simplă formalitate !

e)

În secvența dată, se vede foarte clar faptul că prima ramură "altfel" corespunde lui "dacă $b \geq c$ ", iar al doilea "altfel" se asociază cu "dacă $a > b + c$ ". Să vedem cum ar arăta secvența fără cuvintele cheie "inceput" și "sfârșit":

```
citește a,b,c;
dacă (a>b+c) atunci
  dacă (b>=c) atunci
    x←(a+b)/(2*c);
  altfel
    x←0;
altfel
  x←(a/c+b/c)/2;
scrie x;
```

Conform regulii prezentată pe larg în partea de teorie, într-un lanț de instrucțiuni "dacă-altfel" imbricate, în mod implicit, în absența unor cuvinte cheie "inceput" și "sfârșit", fiecare ramură "altfel" se asociază primului "dacă" de dinaintea lui care nu a fost încă asociat. Altfel spus, asocierile trebuie stabilite dinspre interiorul înspre exteriorul secvenței. Mai exact:

– pentru primul "altfel", cea mai apropiată ramură "dacă" situată înaintea sa este "dacă $b \geq c$ ";

– pentru al doilea "altfel", căutăm de jos în sus primul "dacă" de dinaintea sa care nu a fost încă asociat, și ajungem astfel la "dacă $a > b + c$ ".

Observăm că a rezultat aceeași cuplare a perechilor "dacă-altfel" pe care o aveam în secvența inițială, cea care conținea și cuvintele "inceput" și "sfârșit". Pe cale de consecință, aceste cuvinte cheie erau complet inutile, iar eliminarea lor nu modifică cu nimic efectul execuției algoritmului.

Încercați singuri ! - Tema nr. T.2.

►1. Se dă secvența de algoritm de mai jos:

```
citește a,b,c; dacă (a>b+c) atunci x←(a/c+b/c)/2;
altfel inceput dacă (b>=c) atunci x←(a+b)/(2*c);
altfel x←0; sfarsit scrie x;
```

a) Rescrieți secvența, folosind o așezare în pagină mai "aspectuasă", cu liniile de algoritm scrise una sub alta, și aliniate astfel încât să se vadă fiecare ramură "altfel" cărei ramuri "dacă" îi corespunde.

b) Deduceți ce se afișează în urma execuției secvenței, dacă valorile citite de la tastatură sunt $a=2$, $b=5$ și $c=7$.

c) Găsiți un set de valori pentru variabilele a , b și c , astfel încât în urma executării secvenței să se afișeze valoarea 0.

d) Rescrieți algoritmul într-un mod mai scurt, folosind un număr mai mic de ramuri de tip "dacă-altfel".

e) Dacă din algoritm am șterge cuvintele "inceput" și "sfârșit", efectul execuției algoritmului s-ar schimba, sau ar rămâne același ? Justificați răspunsul.

►2. Fie următoarea secvență descrisă în pseudocod:

a) Ce valori se vor afișa în urma executării secvenței, în cazul în care de la tastatură se citesc valorile $k=0$, $i=5$ și $j=6$?

b) Dați un exemplu de valori pe care le-ar putea primi prin citire variabilele de intrare i , j și k , astfel încât, în urma executării secvenței, să se afișeze două valori egale între ele.

c) Rescrieți algoritmul într-un mod mai scurt, folosind un număr mai mic de ramuri de tip "dacă-altfel".

```
citește i,j,k;
dacă (k>0) atunci
  dacă (i#j) atunci
    x←0;
  altfel
    x←1;
  altfel
    x←2;
  dacă (i#j) atunci
    dacă (k>0) atunci
      y←0;
    altfel
      y←2;
  altfel
    y←1;
scrie x,y;
```

►3. Fie următorul algoritm descris în pseudocod, în care variabilele x , y și z sunt numere reale.

```
citește x,y,z;
dacă (y+z==0 SAU x+z==0 SAU x+y==0) atunci
  scrie "FALS";
altfel
  inceput
    E←x/(z+y)+y/(x+z)+z/(x+y);
    dacă (2*E<3) atunci
      scrie "FALS";
    altfel
      scrie "ADEVARAT"
  sfârșit
```

a) Ce se va afișa pe ecran, dacă de la tastatură se introduc valorile $a=20$, $b=55$, $c=6$?

b) Găsiți un set de valori pentru variabilele a , b și c astfel încât, în urma executării secvenței, să se afișeze textul "FALS".

c) Rescrieți secvența dată, inversând între ele corpurile ramurilor "dacă" și "altfel" în cadrul primeia dintre cele două instrucțiuni de selecție simplă (cu modificarea corespunzătoare a condiției).

Aplicație R.II.3. Funcție matematică

Să se calculeze valoarea funcției matematice $f(x)$, pentru o valoare a lui x introdusă de la tastatură:

$$f: \mathbb{R} \rightarrow \mathbb{R}, f(x) = \begin{cases} x^2 + 1, & \text{pentru } x \leq -3 \\ x - 2, & \text{pentru } x \in (-3, 3) \\ x^2 - 4x + 5, & \text{pentru } x \geq 3 \end{cases}$$

Rezolvare

Nu putem introduce în calculator o funcție algebrică așa cum o cunoaștem din matematică, cu argument, domeniu, codomeniu și lege de asociere. Vom proceda în felul următor: argumentul x al funcției va fi definit ca o variabilă, iar valoarea sa se va citi de la tastatură; conform valorii citite pentru x , trebuie aleasă una dintre cele trei "ramuri" ale funcției, adică una dintre expresiile x^2+1 , $x-2$ și x^2-4x+5 ; pentru valoarea respectivă a lui x , expresia aleasă se evaluează, obținându-se o valoare pe care o atribuim unei variabile E .

De exemplu, pentru $x=1$ se va alege ramura din mijloc a funcției. Rezultă $E=x-2$, adică $E=1-2=-1$.

În concluzie, după citirea lui x urmează un șir de testări:

- dacă $x \leq -3$, atribuim lui E valoarea expresiei $x^2 + 1$;
- în caz contrar:
- ✓ dacă $x \in (-3, 3)$ atribuim lui E valoarea expresiei $x - 2$ (e suficient să se verifice dacă $x < 3$);
- ✓ în caz contrar, nu mai trebuie testat nimic (deoarece nu mai rămâne decât posibilitatea $x \geq 3$) și atribuim lui E valoarea expresiei $x^2 - 4 \cdot x + 5$.

```

citește x;
dacă (x<=-3) atunci
    E ← x*x+1;
altfel
    dacă (x<3) atunci
        E ← x-2;
    altfel
        E ← x*x-4*x+5;
scrie E;
    
```

Încercați singuri! - Tema nr. T.3.

►1. Pentru fiecare dintre funcțiile matematice de mai jos, să se scrie câte un program care afișează valoarea funcției pentru o valoare a argumentului x citită de la tastatură:

$$a) f(x) = \begin{cases} x + 1, & \text{dacă } 0 < x < 1 \\ x^2 - 5 \cdot x + 6, & \text{altfel} \end{cases}$$

$$b) f(x) = \begin{cases} \sqrt{x^2 + x + 2}, & \text{pentru } x < 1 \\ 3x - 1, & \text{pentru } 1 \leq x \leq 2 \\ \frac{5x - 2}{x + 2}, & \text{pentru } x > 2 \end{cases}$$

Aplicație R.II.4. Câtul și restul

(Bacalaureat iulie 2008, varianta 96)

Realizați un algoritm care citește de la tastatură două numere naturale distincte, apoi afișează câtul și restul împărțirii celui mai mare dintre cele două numere la cel mai mic dintre ele. Dacă numerele sunt egale, sau dacă împărțirea nu se poate efectua, se va afișa mesajul "Eroare".

Rezolvare

Algoritmul începe cu citirea de la tastatură a numerelor a și b . Prima condiție care trebuie testată este aceea ca numerele să fie distincte, deci:

- dacă cele două valori sunt egale, atunci scrie mesajul "Eroare";
- în caz contrar:
 - ✓ dacă a este mai mare ca b , atunci trebuie afișat câtul împărțirii lui a la b , dar pentru aceasta este necesar ca împărțitorul să fie nenul, deci:
 - dacă b este diferit de 0, atunci scrie pe ecran câtul și restul împărțirii lui a la b , adică a/b și $a \% b$, separate printr-un spațiu (vă reamintim că, în cazul în care operanzii a și b sunt ambii întregi, expresia " a/b " furnizează câtul întreg al împărțirii lui a la b , adică fără parte zecimală);
 - în caz contrar, scrie de asemenea mesajul "Eroare";
 - ✓ dacă a este mai mic ca b , atunci trebuie afișat câtul împărțirii lui a la b , și urmează o testare similară cu cea de dinainte:
 - dacă a este diferit de 0, atunci scrie pe ecran câtul și restul împărțirii lui b la a , separate printr-un spațiu;
 - în caz contrar, trebuie din nou afișat mesajul "Eroare";

Mai precizăm că ramura "altfel" a primei instrucțiuni de selecție "dacă-altfel" conține alte două instrucțiuni "dacă-altfel" (cu condițiile "dacă $a > b$ " și "dacă $b > a$ "), motiv pentru care trebuie cuprins între cuvintele cheie "început" și "sfârșit". În absența acestor cuvinte, calculatorul va considera că pe ramura "altfel" a primului "dacă-altfel" se găsește doar următoarea instrucțiune, adică numai prima dintre cele două secvențe "dacă-altfel" care urmează (cel cu condiția "dacă $a > b$ "), iar următorul "dacă-altfel" (cel cu condiția "dacă $b > a$ ") va fi scos în afara selecției (reamintim în acest punct de vedere că o secvență "dacă-altfel" împreună cu cele două ramuri și corpurile lor, este privită ca o singură instrucțiune).

Nu în ultimul rând trebuie remarcat că în loc de "dacă $b > a$ " puteam să scriem pur și simplu "altfel". De ce? Cazul contrar lui " $a > b$ " este " $a \leq b$ ", dar situația " $a = b$ " a fost tratată separat mai sus, deci rămâne doar " $a < b$ ", adică exact ceea ce testăm în respectiva linie "dacă". Am preferat această variantă, pentru că în cealaltă am fi ajuns la un lanț de imbricări ale structurilor "dacă-altfel" cu trei nivele, care ar fi făcut destul

de anevoioasă înțelegerea algoritmului. Celor care au pătruns deja profund în tainele structurilor de selecție, le propunem să se încumete să abordeze și cealaltă versiune !

```
citește a,b;
dacă (a==b) atunci
    scrie 'Eroare';
altfel
    inceput
        dacă (a>b) atunci
            dacă (b≠0) atunci
                scrie a/b, ' ', a%b;
            altfel
                scrie "Eroare";
        dacă (b>a) atunci
            dacă (a≠0) atunci
                scrie b/a, ' ', b%a;
            altfel
                scrie "Eroare";
    sfârșit
```

Încercați singuri ! - Tema nr. T.4.

► 1. Fiind date trei numere naturale nenule a , b și k , să se verifice dacă fracția a/b se simplifică prin k . În caz afirmativ se va afișa și fracția simplificată. Se va testa și condiția ca numitorul fracției să fie nenul.

► 2. Se citește de la tastatură un număr natural care reprezintă un an calendaristic. Realizați un algoritm care testează dacă anul este bisect sau nu, afișând un mesaj corespunzător. Un an este bisect dacă:

- (este divizibil cu 4 și nu este divizibil cu 100) SAU
- este divizibil cu 400.

Aplicație R.II.5. Laturi în triunghi

Se citesc de la tastatură trei numere reale a , b , c . Să se verifice dacă aceste numere pot fi laturile unui triunghi și în caz afirmativ să se determine natura triunghiului în funcție de laturi (echilateral, isoscel, oarecare). Trei numere pot fi laturile unui triunghi, dacă sunt pozitive și suma a oricare două dintre ele este mai mare decât al treilea.

Exemplu: Numerele $a=4$, $b=4$ și $c=7$ sunt laturile unui triunghi isoscel, iar valorile $a=2$, $b=3$ și $c=9$ nu pot fi laturile unui triunghi (din cauză că $2+3<9$).

Rezolvare

 Algoritm ilustrează funcționarea structurilor selective imbricate. Citim numerele a , b , c , apoi testăm dacă îndeplinesc condiția de a fi laturile unui triunghi: $(a>0 \text{ și } b>0 \text{ și } c>0 \text{ și } a+b>c \text{ și } a+c>b \text{ și } b+c>a)$

 În cazul în care această condiție este adevărată, printr-o altă structură de selecție determinăm natura triunghiului, afișând un mesaj:

- dacă triunghiul are perechi de laturi egale, adică $(a==b \text{ și } a==c)$, atunci este echilateral (automat rezultă și că b este egal cu c , conform tranzitivității relației de egalitate);
- în caz contrar, triunghiul poate fi isoscel sau oarecare:
 - dacă două laturi sunt egale, $(a==b \text{ SAU } a==c \text{ SAU } b==c)$, atunci este isoscel;
 - în cazul contrar și celui anterior nu mai testăm nimic, pentru că triunghiul nu mai poate fi decât oarecare.

```
citește a,b,c;
dacă (a>0 și b>0 și c>0 și a+b>c și a+c>b și b+c>a) atunci
    inceput
        scrie "Numerele sunt laturi";
        dacă (a==b și b==c) atunci
            scrie "Triunghi echilateral ";
        altfel
            dacă (a==b SAU a==c SAU b==c) atunci
                scrie "Triunghi isoscel ";
            altfel
                scrie "Triunghi oarecare ";
    sfârșit
altfel
    scrie "Nu e triunghi ";
```

 Observați ce conține corpul ramurii "dacă" aferent primei condiții: o instrucțiune de afișare și o altă instrucțiune de selecție "dacă-altfel". Din acest motiv, întreg corpul în cauză este cuprins între cuvintele "inceput" și "sfârșit".

Încercați singuri ! - Tema nr. T.5.

► 1. Se citesc de la tastatură trei numere reale a , b , c . Scrieți pseudocodul unui algoritm care verifică dacă acestea pot fi unghiuri ale unui triunghi exprimate în grade (sunt pozitive și suma lor este 180), iar în caz afirmativ să se determine natura triunghiului (optuz-unghic, dreptunghic sau ascuțit-unghic).

► 2. Se citesc trei numere întregi a , b , c . Scrieți pseudocodul unui algoritm care verifică dacă aceste numere sunt în progresie aritmetică (trei numere sunt în progresie aritmetică dacă unul din ele este media aritmetică a celorlalte două), afișând un mesaj sugestiv.

Aplicație R.II.6. Rezolvarea ecuației de gradul II

Scrieți un algoritm pentru rezolvarea ecuației de gradul II de forma $ax^2+bx+c=0$, unde coeficienții a , b , c se citesc de la tastatură. Se vor trata toate cazurile posibile.

Rezolvare

Reamintim: $\Delta = b^2 - 4ac$, $x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$. Pașii algoritmului sunt următorii:

- citim cei trei coeficienți ai ecuației în variabilele a, b, c ;
- rezolvăm ecuația de gradul II cu coeficienții a, b, c citiți anterior, tratând toate cazurile:
 - dacă a este egal cu 0:
 - dacă b este egal cu 0, atunci: dacă și c este egal cu 0, atunci ecuația are o infinitate de soluții, iar dacă c este diferit de 0 ecuația este una imposibilă; în ambele situații nu se rezolvă ecuația, tipărindu-se un mesaj, motiv pentru care ele au fost comasate.
 - în caz contrar, ecuația devine $bx + c = 0$ cu rădăcina $x \leftarrow -c/b$, care se afișează;
 - în caz contrar: se calculează discriminantul: $\text{delta} \leftarrow b^2 - 4*a*c$ și:
 - dacă $\text{delta} < 0$, ecuația nu are soluții și se afișează un mesaj;
 - în caz contrar ($\text{delta} \geq 0$), calculăm rădăcinile

$$x1 \leftarrow (-b - \sqrt{\text{delta}}) / (2*a),$$

$$x2 \leftarrow (b + \sqrt{\text{delta}}) / (2*a)$$
 pe care le afișăm.

Observație: Modul în care introducem radicalul în program va fi prezentat în capitolul următor. La nivel de pseudocod folosim simbolul cunoscut din matematică.

```

citește a,b,c;
dacă (a==0) atunci
    dacă (b==0) atunci
        scrie "Nu se rezolva ecuatia";
    altfel
        scrie -c/b;
altfel
    început
        delta ← b*b-4*a*c;
        dacă (delta<0) atunci
            scrie "Nu are solutii reale";
        altfel
            început
                x1 ← (-b+√delta)/(2*a);
                x2 ← (-b-√delta)/(2*a);
                scrie x1,x2;
            sfârșit
        sfârșit
    sfârșit

```

Încercați singuri! - Tema nr. T.6.

►1. Realizați un algoritm pentru rezolvarea unui sistem de două ecuații cu două necunoscute, de forma:

$$\begin{aligned} a \cdot x + b &= m \\ c \cdot x + d &= n \end{aligned}$$

Coeficienții a, b, c, d, m, n , se citesc de la tastatură. Se va testa și condiția de existență a sistemului, " $a \neq 0$ ȘI $c \neq 0$ ". Se cer două variante, aferente celor două metode de rezolvare cunoscute: metoda reducerii și metoda substituției.

Ați înțeles? Probleme cu răspuns scurt

Următorii trei itemi se referă la secvența de instrucțiuni de mai jos, în care toate variabilele sunt de tip întreg.

```

citește a,b,c;
x←1;
dacă (not(a<=b) SAU not(a<=c))
    început
        dacă (b>=c) atunci
            scrie -x;
        sfârșit
    altfel
        dacă (b<c) atunci
            scrie x;

```

► 1. Ce valoare se va afișa pe ecran, dacă de la tastatură se citesc valorile $a=2$, $b=3$, $c=4$?

► 2. Precizați un set de valori care ar putea fi citite pentru variabilele a, b și c , astfel încât să se afișeze valoarea -1.

► 3. Rescrieți condiția din prima linie "dacă", înlocuind-o cu o expresie logică echivalentă, astfel încât să nu fie folosit operatorul logic de negație.

► 4. (Bacalaureat iulie 2008, varianta 44)

Care sunt valorile variabilelor a și b după executarea secvenței de mai jos, dacă inițial ele aveau valori diferite?

```

dacă (a<b SAU a>b) atunci
    început
        a←1;
        b←a;
    sfârșit
dacă (a==b) atunci
    b←0;

```

► 5. (Bacalaureat iulie 2009, varianta 91)

Știind că variabila întreagă nr memorează valoarea 5, stabiliți ce mesaj se va afișa în urma execuției secvenței de program de mai jos:

```

dacă (nr<7) atunci
    dacă (nr<3) atunci
        scrie "BINE";
    altfel
        scrie "FOARTE BINE";
altfel
    scrie "RAU";

```

- a) BINE RĂU b) FOARTE BINE c) RAU d) BINE

► 6. (Bacalaureat iulie 2009, varianta 90)

Care dintre următoarele instrucțiuni atribuie variabilei întregi t valoarea -1 dacă și numai dacă variabilele întregi a și b sunt nenule și au semne diferite?

- a) dacă $(a > 0 \text{ SAU } b < 0)$ atunci $t \leftarrow -1$;
- b) dacă $(a > 0 \text{ ȘI } b < 0)$ atunci $t \leftarrow -1$;
- c) dacă $(a * b < 0)$ atunci $t \leftarrow -1$;
- d) dacă $(a * b > 0)$ atunci $t \leftarrow -1$;

► 7. (Bacalaureat iulie 2008, varianta 75)

Care dintre următoarele secvențe de instrucțiuni în pseudocod testează dacă valoarea variabilei z este cea mai mare dintre valorile variabilelor x , y și z , iar în caz afirmativ o afișează?

- a) dacă $(x < y \text{ ȘI } y < z)$ atunci scrie z ;
- b) dacă $(x < y \text{ ȘI } x < z)$ atunci scrie z ;
- c) dacă $(x < y \text{ SAU } y < z \text{ SAU } x < z)$ atunci scrie z ;
- d) dacă $(x > z \text{ SAU } y > z)$ atunci scrie z ;



Probleme propuse

Tema nr. T.R.1.

► 1. Să se rezolve ecuația de gradul I de forma $ax + b = 0$, unde coeficienții a și b se citesc de la tastatură. Se va trata și cazul particular $a = 0$.

► 2. Se citește de la tastatură măsura unui unghi x exprimat în grade. Să se verifice dacă unghiul aparține primului cadran al cercului trigonometric ($x \in (0, 90^\circ)$), iar în caz afirmativ să se afișeze măsura sa exprimată în radiani.

► 3. (Bacalaureat iulie 2008, varianta 99)

Scrieți un algoritm care citește de la tastatură trei valori distincte notate cu a , b și c , apoi afișează una dintre următoarele valori:

- 1, dacă $a < b$ și $b > c$;
- 2, dacă $a > b$ și $b < c$;
- 3, dacă b este cea mai mare dintre valori;
- 4, dacă b este cea mai mică dintre valori;
- 5, în toate celelalte cazuri posibile.

► 4. Să se verifice dacă trei numere naturale date a , b și c sunt pitagorice (adică dacă verifică una dintre condițiile $a^2 = b^2 + c^2$, $b^2 = a^2 + c^2$, $c^2 = a^2 + b^2$).

► 5. Se citește de la tastatură o literă. Se cere să se verifice dacă ea este vocală sau consoană (considerăm alfabetul englez în care vocalele sunt 'a', 'e', 'i', 'o', 'u').

► 6. Să se verifice dacă un număr natural dat este pătrat perfect, afișându-se un mesaj corespunzător.

Indicații. Un număr natural x este pătrat perfect dacă radical din x este egal cu parte întreagă din radical din x . Exemplu: Numărul 4 este pătrat perfect ($\sqrt{4} = 2$ și $\lfloor \sqrt{4} \rfloor = \lfloor 2 \rfloor = 2$), iar 5 nu este ($\sqrt{5} = 2.23 \dots$ $\lfloor \sqrt{5} \rfloor = \lfloor 2.23 \dots \rfloor = 2$).

► 7. Fiind date trei numere reale a , b și c citite de la tastatură, să se afișeze în ordine crescătoare.

► 8. Fiind date trei numere reale a , b și c citite de la tastatură, să se afișeze în ordine descrescătoare.

► 9. Cunoscând măsurile a două unghiuri exprimate în grade, minute și secunde, să se alcătuiască un algoritm care afișează măsura unghiului sumă exprimat în același mod. Exemplu: $23^\circ 34' 55'' + 22^\circ 33' 20'' = 46^\circ 8' 15''$.

Indicații. Știm că un minut are 60 de secunde și un grad are 60 de minute. Se adună mai întâi secunde, apoi minutele și în final gradele. Dacă suma secundelor depășește 60, am obținut un minut care se propagă către adunarea minutelor sub formă de "transport", iar ceea ce rămâne peste 60 reprezintă numărul de secunde ale rezultatului. La fel se întâmplă și la adunarea minutelor, putând apărea un grad transport către suma gradelor. În exemplul din enunț: $55'' + 20'' = 75''$ adică $15''$ și un minut transport; $34' + 33' + 1' = 68'$ adică $8'$ și un grad transport; $23^\circ + 22^\circ + 1^\circ = 46^\circ$.

► 10. Se citesc de la tastatură două numere întregi x și y . Scrieți un algoritm care calculează valoarea expresiei:

- a) $E(x, y) = \begin{cases} \max(x, y) & , \text{ dacă } x^2 - y^2 > 1 \\ \min(x, y) & , \text{ în caz contrar} \end{cases}$
- b) $f(x) = \max(x^2 + x, 4x - 2)$



II.3.2. Structura de selecție multiplă

Am văzut că prin intermediul instrucțiunii de selecție "dacă-atunci-altfel", putem alege o secvență de instrucțiuni din două posibile, în funcție de valoarea de adevăr a unei condiții. De multe ori apare însă necesitatea de a alege o secvență din mai multe posibile. Această alegere se va face în funcție de valoarea unei expresii numită *selector*. În cele mai multe cazuri expresia-selector este o variabilă.

Valorile selectorului în funcție de care alegem secvența se vor numi *cazuri*.

Așadar:

Linia "**alege** (<selector>)" se traduce prin "alege unul din mai multe cazuri posibile, în funcție de valoarea variabilei (<selector>". În limbajul C++, numele variabilei selector trebuie scris între paranteze rotunde. Pentru a vă obișnui încă de pe acum cu această regulă de sintaxă, o vom adopta și în pseudocod;

Fiecare dintre simbolurile $\langle \text{val}_1 \rangle$, $\langle \text{val}_2 \rangle$, ..., $\langle \text{val}_n \rangle$, reprezintă o valoare posibilă a selectorului $\langle \text{selector} \rangle$, sau un set de valori ale acestuia separate prin virgulă. Astfel, de exemplu, linia "cazul $\langle \text{val}_1 \rangle$:" se traduce prin "cazul în care selectorul are valoarea sau setul de valori simbolizat(ă) prin $\langle \text{val}_1 \rangle$ ";

Fiecărui caz îi corespunde o secvență de instrucțiuni, iar calculatorul va alege unul dintre cazuri executând secvența aferentă în funcție de valoarea selectorului. Astfel, în cazul în care selectorul are valoarea / setul de valori $\langle \text{val}_1 \rangle$ se execută secvența de instrucțiuni $\langle \text{secvența}_1 \rangle$, dacă selectorul are valoarea / setul de valori $\langle \text{val}_2 \rangle$ are loc execuția secvenței $\langle \text{secvența}_2 \rangle$, ș.a.m.d.. Observați că pe fiecare "ramură", după secvența de instrucțiuni aferentă, urmează cuvântul cheie "ieșire". Acest fapt se datorează modului în care funcționează instrucțiunea de selecție multiplă în limbajul C++; în absența instrucțiunii "ieșire", compilatorul ar trece la analizarea următoarelor "cazuri", fiind posibil ca în mod accidental valoarea pe care o are selectorul în acel moment să determine și execuția altor "ramuri" ale instrucțiunii de selecție multiplă. În felul acesta ar fi alterat sensul instrucțiunii de selecție multiplă, acela de a alege *un singur caz* din mai multe posibile. Prezența cuvântului-cheie "ieșire" la sfârșitul fiecăruia dintre "cazuri", va determina compilatorul să abandoneze complet instrucțiunea de selecție multiplă după execuția secvenței de instrucțiuni aferentă "cazului" selectat. Evident, tot "corpul" fiecărei "ramuri" trebuie cuprins între cuvintele cheie "inceput" și "sfârșit" (chiar dacă secvența de instrucțiuni aferentă "ramurii" este alcătuită dintr-o singură instrucțiune, împreună cu instrucțiunea "ieșire" sunt deja două instrucțiuni !);

Pe lângă "ramurile" ce descriu cazurile, mai poate exista o ramură numită "altfel". Aceasta se va executa dacă valoarea selectorului nu este cuprinsă în nici unul dintre cazurile anterioare $\langle \text{val}_1 \rangle$, $\langle \text{val}_2 \rangle$, ..., $\langle \text{val}_n \rangle$. Menționăm însă că prezența ramurii "altfel" nu este obligatorie, situație în care dacă nu este îndeplinit nici unul din cazurile date, pur și simplu nu se va executa nimic.

```

alege (<selector>)
inceput
cazul <val_1>:
    inceput
    <secvența_1>
    ieșire
sfârșit
cazul <val_2>:
    inceput
    <secvența_2>
    ieșire
    sfârșit
.....
cazul <val_n>:
    inceput
    <secvența_n>
    ieșire
    sfârșit
altfel
    inceput
    <secvența_0>
    ieșire
    sfârșit
sfârșit

```

Întreaga instrucțiune de selecție multiplă trebuie să se încheie cu cuvântul "sfârșit".

Exemplu:

```

citește x;
alege x dintre
-1: y ← x+2;
0: inceput
    scrie "Ati introdus 0";
    y ← x;
    sfârșit
1: y ← x-1;
altfel
    inceput
    scrie "Nu e -1,0,1 ";
    y ← 100;
    sfârșit
sfârșit
scrie y;

```

Se citește de la tastatură valoarea variabilei x , care este selectorul unei instrucțiuni de selecție multiple. În funcție de valoarea citită a lui x , se alege unul din cele patru cazuri: dacă s-a citit $x=-1$, atunci se face atribuirea $y \leftarrow x+2$; dacă s-a citit $x=0$, atunci se execută o secvență de două instrucțiuni cuprinsă între cuvintele "inceput" și "sfârșit": afișarea unui mesaj și atribuirea $y \leftarrow x$. dacă s-a citit $x=1$, atunci are loc atribuirea $y \leftarrow x-1$; în sfârșit, dacă valoarea citită a lui x este diferită de $-1, 0, 1$, atunci se intră pe ramura "altfel", care conține o instrucțiune de afișare și atribuirea $y \leftarrow 100$. După cuvântul "sfârșit" care marchează încheierea instrucțiunii de selecție multiplă, urmează afișarea variabilei y .

Aplicație R.II.7. Alegere

Să se alcătuiască un program care citește două numere reale a și b , apoi afișează media aritmetică, suma pătratelor sau suma cuburilor celor două numere, în funcție de dorința utilizatorului.

Rezolvare

☞ Fie o variabilă de selecție opt , în funcție de valoarea căreia calculatorul va alege opțiunea utilizatorului. Anumite valori ale variabilei de selecție corespund acțiunilor posibile. Facem convenția că aceste valori sunt 1, 2, 3 (evident pot fi alese orice valori).

☞ Citim de la tastatură numerele a , b și valoarea variabilei opt . Apoi, cerem calculatorului să testeze valoarea lui opt într-o instrucțiune de selecție multiplă și să aleagă în funcție de această valoare acțiunea pe care o va executa în continuare:

- dacă **opt** a primit valoarea 1 va calcula și va afișa media aritmetică;
- dacă valoarea citită a lui **opt** a fost 2, va calcula suma pătratelor celor două numere;
- pentru **opt=3** va afla suma cuburilor;
- pentru orice altă valoare a lui **opt** în afară de 1, 2, 3, va tipări un mesaj de eroare (aceasta este ramura "altfel" a instrucțiunii de selecție).

📖 Pentru fiecare dintre cele trei mărimi calculate, există două posibilități:

- să o memorăm prin atribuire într-o variabilă și apoi să afișăm variabila;
- să o afișăm direct ca expresie.

```

citește a,b;
citește opt;
alege opt dintre
1: început
    ma←(a+b)/2;
    scrie ma;
sfârșit
2: început
    sp←a*a+b*b;
    scrie sp;
sfârșit
3: început
    sc←a*a*a+b*b*b;
    scrie sc;
sfârșit
altfel scrie 'Eroare';
sfârșit

```

```

citește a,b;
citește opt;
alege opt dintre
1: scrie (a+b)/2;
2: scrie a*a+b*b;
3: scrie a*a*a+b*b*b
altfel
    scrie 'Eroare';
sfârșit

```

📌 Încercați singuri! - Tema nr. T.7.

►1. Scrieți un algoritm care, în funcție de dorința utilizatorului, calculează și afișează: aria unui pătrat de latură **L**, sau aria unui cerc de rază **r**, sau aria unui triunghi cu baza **b** și înălțimea **h**.



Probleme propuse

Tema nr. T.R.2.

►1. Se citește de la tastatură un număr reprezentând o zi calendaristică dintr-o lună oarecare. Să se afișeze decada căreia îi aparține ziua.

►2. Se citește de la tastatură o cifră între 1 și 7 reprezentând o zi din săptămână. Să se afișeze denumirea zilei. Zilele săptămânii sunt numerotate astfel: 1→ luni, 2→ marți, ..., 7→ duminică.

►3. Scrieți un program care simulează funcționarea unui calculator de buzunar: calculatorul "va citi" două numere reale și unul din caracterele '+', '-', '*',

'/' reprezentând operațiile aritmetice fundamentale, va efectua operația dată de caracterul citit, folosind drept operanzi numerele citite și va tipări rezultatul.

►4. O persoană dorește să facă o excursie în Grecia, pe litoralul Mării Mediterane, prin intermediul unei agenții de turism. Prețul excursiei variază în funcție de sezon, astfel: 150 Euro între 22 decembrie și 20 martie, 200 Euro între 21 martie și 20 iunie, 300 Euro între 21 iunie și 10 septembrie, și 250 Euro între 11 septembrie și 21 decembrie. Cunoscând data pe care o alege persoana pentru a pleca în excursia respectivă (citită de la tastatură prin intermediul a două variabile reprezentând ziua și luna), să se afișeze prețul excursiei.



II.4. Structuri repetitive

II.4.1. Structura repetitivă cu test inițial

Așa cum în matematică o problemă poate avea mai multe rezolvări, la fel un algoritm poate fi proiectat în mai multe moduri. Apare aici următoarea întrebare: care dintre algoritmii aceleiași probleme este mai eficient?

Să luăm ca exemplu următoarea problemă:

📌 **Exemplu:** Se citesc trei numere întregi. Să se afișeze suma lor.

Algoritmul este foarte simplu. Citim de la tastatură trei numere întregi **x**, **y**, **z**, apoi calculăm suma lor **S=x+y+z**.

```

citește x,y,z;
S←x+y+z;
scrie S;

```

Pe cât de simplu, pe atât de ineficient, din cauza ... risipei de variabile. Nu există altă soluție? Ba da, întrucât enunțul problemei nu ne obligă să adunăm toate numerele "deodată". Putem să le adunăm pe rând. Cum anume? Citindu-le pe rând în aceeași variabilă **x** și adăugându-le tot pe rând la **S**.

📌 **Exemplu:** Să presupunem că cele trei numere sunt 11, 12, 13.

Atribuim lui **S** valoarea 0. Aceasta va fi valoarea inițială a lui **S**;

Citim primul număr 11 în variabila **x**, apoi îl adăugăm la valoarea inițială a lui **S**. Cum facem acest lucru? Printr-o atribuire care la prima vedere s-ar putea să vi se pară foarte ciudată: **S←S+x**;

Vom spune: "adaugă valoarea lui **x** la valoarea pe care o are **S** în acel moment și memorează rezultatul tot în **S**", sau "noul **S** va fi egal cu vechiul **S** plus **x**".

avem **S←S+x**, adică **S=0+11**, **S=11**.

Citim al doilea număr 12 tot în variabila **x**, apoi îl adăugăm la valoarea curentă a lui **S**,

tot prin atribuirea $S \leftarrow S+x$. Se va obține $S=11+12$, adică $S=23$, care este tocmai suma primelor două numere.

În continuare procedăm analog: citește $x=13$, apoi $S \leftarrow S+x$, deci $S=23+13$, $S=36$

Această ultimă valoare a lui S este chiar suma cerută, pe care o afișăm.

```
S ← 0;
citește x;
S ← S+x;
citește x;
S ← S+x;
citește x;
S ← S+x;
scrie S;
```

Cu numai două variabile am obținut un algoritm mult mai eficient din acest punct de vedere, dar în care se poate observa cu ușurință un alt inconvenient: aceeași secvență de două instrucțiuni, alcătuită din $\{ \text{citește } x; \}$ și $\{ S \leftarrow S+x; \}$ se repetă de trei ori ! Imaginați-vă că dacă avem de calculat suma a 100 de numere, suntem nevoiți să scriem aceeași secvență de o sută de ori, ceea ce, trebuie să recunoașteți, nu e tocmai plăcut. Care e soluția?

Vom proiecta o structură în care scriem secvența respectivă o singură dată și comandăm execuția ei de câte ori dorim. O astfel de structură se numește **structură repetitivă** sau **ciclu**.

Exemplu:

Revenind la problema anterioară, cum putem proiecta un ciclu în care secvența de instrucțiuni $\{ \text{citește } x; S \leftarrow S+x; \}$, scrisă o singură dată, să se execute de trei ori ? În primul rând avem nevoie de o variabilă, notată în continuare cu i , care să numere execuțiile. Mai exact, valoarea lui i va indica la a câta execuție a secvenței ne aflăm. Astfel, inițial, la prima execuție valoarea lui i va fi 1, apoi la a doua execuție va fi 2 și în sfârșit la a treia execuție vom avea $i=3$.

Fiecare execuție a secvenței va constitui un **pas al ciclului**. Dar la sfârșitul fiecărui pas al ciclului, după execuția celor două instrucțiuni din secvență, calculatorul trebuie să mai facă încă ceva. Ce anume ? Să mărească cu 1 valoarea lui i pentru a marca faptul că urmează să treacă la pasul următor. Cum are loc mărirea valorii lui i cu 1 ? Prin atribuirea $i \leftarrow i+1$. Deși comparativ cu ceea ce știți din matematică această atribuire pare o ciudățenie, ea este nu numai corectă ci și foarte des folosită. O traducem astfel: "mărește valoarea curentă a lui i cu 1, și memorează rezultatul tot în i ". Astfel, de exemplu, inițial $i=1$, apoi la finele primului pas, prin execuția atribuirii $i \leftarrow i+1$ se va obține $i=i+1$ adică $i=2$, apoi după al doilea pas vom avea $i=2+1$ adică $i=3$, ș.a.m.d. Operația se numește **incrementarea lui i cu 1**.

Am lămurit așadar faptul că la fiecare pas al ciclului se vor executa trei instrucțiuni: cele două instrucțiuni ale secvenței care trebuie să se repete, plus incrementarea lui i .

```
citește x;
S ← S+x;
i ← i+1;
```

Prin urmare, plecând de la valoarea inițială 1, contorul i se mărește la fiecare pas cu o unitate. Dar acest lucru se va petrece la infinit ? Nu ! Procesul trebuie să se oprească în momentul în care variabila i a ajuns la valoarea 3, pentru că s-au "consumat" exact trei execuții ale secvenței. Altfel spus, ciclul se execută cât timp valoarea lui i este mai mică sau egală cu 3. Aceasta este așa-numita **condiție de continuare a ciclului**. În pseudocod ea se scrie astfel:

cât timp ($i \leq 3$) execută

Din dorința de a respecta o cerință a limbajului C++ în materie de sintaxă, în pseudocod vom scrie între paranteze expresia logică aferentă condiției din linia "cât timp" (în cazul nostrum " $i \leq 3$ ").

După această linie trebuie să urmeze firește corpul ciclului, adică secvența de instrucțiuni care se execută la fiecare pas. Respectând convenția de la instrucțiunile de selecție, vom cuprinde corpul ciclului între cuvintele "început" și "sfârșit" deoarece conține mai mult de o instrucțiune. După cuvântul "sfârșit" care marchează încheierea ciclului, mai urmează o singură instrucțiune, afișarea valorii finale a lui S .

Prin urmare pseudocodul acestei probleme poate fi scris în felul următor:

```
S ← 0; i ← 1;
cât timp (i ≤ 3) execută
    început
        citește x;
        S ← S+x;
        i ← i+1;
    sfârșit
scrie S;
```

Ca o sistematizare a ceea ce am prezentat până acum, să urmărim pașii ciclului, presupunând că cele trei numere citite de la tastatură pe rând în variabila x sunt 11, 12, 13.

Inițial, $S \leftarrow 0$ și $i \leftarrow 1$;

Pas 1: $i \leq 3$? $1 \leq 3$? da $\Rightarrow$

• citește $x=11$; • $S \leftarrow S+x$, $S=0+11=11$; • $i \leftarrow i+1$, $i=1+1=2$.

Pasul 2: $i \leq 3$? $2 \leq 3$? da $\Rightarrow$

• citește $x=12$; • $S \leftarrow S+x$, $S=11+12=23$; • $i \leftarrow i+1$, $i=2+1=3$.

Pasul 3: $i \leq 3$? $3 \leq 3$? da $\Rightarrow$

• citește $x=13$; • $S \leftarrow S+x$, $S=23+13=36$; • $i \leftarrow i+1$, $i=3+1=4$.

Pasul 4: $i \leq 4$? $4 \leq 3$? nu $\Rightarrow$ iese din ciclu

scrie $S \Rightarrow$ scrie 36.

Observați că practic ciclul a avut trei pași, deoarece corpul său s-a executat de numai trei ori. La pasul al patrulea condiția " $i \leq 3$ " a devenit falsă, fapt care determină ieșirea din ciclu.

Atribuirea $S \leftarrow 0$ dinaintea ciclului se numește **inițializarea lui S** și este foarte importantă. După cum ați văzut, la fiecare pas valoarea lui S se actualizează prin atribuirea $S \leftarrow S+x$, care se citește "noul S va fi vechiul S plus x ". La fiecare pas începând cu al doilea, "vechiul S " este tocmai S -ul obținut la pasul anterior. Dar la primul pas nu putem vorbi de "pasul anterior" ! Din acest motiv inițializarea lui S cu 0 înaintea ciclului este obligatorie, pentru a exista "vechiul S " la primul pas.

Desigur că algoritmul va funcționa și dacă în condiția din linia "cât timp" în loc de 3 avem 4, 5, 6, sau orice altă valoare. Astfel, pornind de la exemplul anterior, putem calcula suma a n numere, unde n poate lua orice valoare.

Exemplu:

Să se calculeze suma a n numere naturale citite pe rând de la tastatură (unde n este cunoscut).

Ar trebui să fie clar faptul că, față de pseudocodul prezentat anterior, apar doar două modificări: valoarea variabilei n trebuie citită de la tastatură înaintea ciclului "cât timp" (poate fi oricât), iar ciclul se execută cât timp $i \leq n$.

```

citește n; // citește numărul de numere n
S ← 0; // inițializează cu 0 variabila S, care în final va fi suma celor n numere
i ← 1; // contorul i numără pașii ciclului; înaintea primului pas inițializăm i cu 1
cât timp (i ≤ n) execută // ciclul se execută cât timp valoarea lui i este ≤ n
    început
    citește x; // citește un număr în variabila x
    S ← S + x; // adaugă numărul citit x la suma S
    i ← i + 1; // incrementează contorul i, marcând trecerea la pasul următor
    sfârșit
scrie S // după ieșirea din ciclu, tipărește suma S a celor n numere
    
```

Mai notăm faptul că ceea ce am prezentat până acum se numește **structură repetitivă cu test inițial**, sau **ciclu cu test inițial**, și că mai există încă două tipuri de structuri repetitive, pe care le veți învăța ceva mai târziu.

Sintaxa: unui ciclu cu test inițial:

```

cât timp (<expr>) execută
    <secv>
    
```

– unde **<expr>** este o condiție (expresie logică), iar **<secv>** este o secvență formată dintr-una sau mai multe instrucțiuni, numită **corpul ciclului**.

Principiul de funcționare:

Atâta timp cât este îndeplinită condiția dată de **<expr>** (cât timp expresia **<expr>** are valoarea 1, corespunzătoare valorii de adevăr "ADEVĂRAT"), execută corpul ciclului **<secv>**:

- 1) se evaluează expresia logică **<expr>**
- 2) ♦ dacă aceasta are valoarea 1, atunci se execută secvența **<secv>**, apoi se revine la pasul "1" (între timp valoarea expresiei **<expr>** s-a putut modifica);
♦ în caz contrar se trece la instrucțiunea de după ciclu.

Observații:

♥ Respectând regula de sintaxă a limbajului C++, vom scrie între paranteze rotunde expresia logică **<expr>** din linia "cât timp";

♥ În cazul în care corpul ciclului conține cel puțin două instrucțiuni,

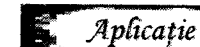
acesta va fi cuprins între cuvintele "început" și "sfârșit".

♥ Testarea condiției are loc la început, deci corpul ciclului poate să nu se execute niciodată.



Algoritmi cu structuri repetitive cu test inițial

Sume și produse cu n termeni



Aplicație R.II.8. Produs de numere naturale impare

Scrieți pseudocodul unui algoritm care calculează produsul numerelor naturale impare mai mici sau egale decât o valoare n dată.

Exemplu: pentru $n=8$ rezultă $P=1 \cdot 3 \cdot 5 \cdot 7=105$, pentru $n=9$ se va calcula $P=1 \cdot 3 \cdot 5 \cdot 7 \cdot 9=945$.



Rezolvare

 Citim valoarea lui n de la tastatură. Calculăm produsul într-un ciclu cu test inițial, folosind o variabilă i , ale cărei valori sunt pe rând termenii sumei, 1, 3, 5, ... Inițial $i \leftarrow 1$ (prima valoare impară), $P \leftarrow 1$;

Atâta timp cât $i \leq n$:

înmulțeste un termen cu produsul P al numerelor impare, și anume valoarea curentă a lui i ($P \leftarrow P \cdot i$, adică "noul P este vechiul P înmulțit cu i ");
variabila i ia următoarea valoare impară ($i \leftarrow i + 2$).

Pentru a exista "vechiul P " la execuția primului pas al ciclului, variabila P trebuie inițializată cu o valoare înainte de ciclu. Valoarea inițială a lui P este 1, ca să nu influențeze rezultatul înmulțirii de la primul pas, știut fiind faptul că înmulțirea cu 1 nu modifică rezultatul ($P \leftarrow P \cdot i$, $P=1 \cdot 1=1$).

```

citește n;
P ← 1;
i ← 1;
cât timp (i ≤ n) execută
    început
    P ← P * i;
    i ← i + 2;
    sfârșit
scrie P;
    
```

Presupunem că s-a citit $n=8$. Inițial $P \leftarrow 1$ și $i \leftarrow 1$. Urmărim pașii ciclului:

Pas 1: $i \leq n$? $1 \leq 8$ da $\Rightarrow P \leftarrow P \cdot i$, $P=1 \cdot 1=1$ • $i \leftarrow i + 2$, $i=1+2=3$
 Pas 2: $i \leq n$? $3 \leq 8$ da $\Rightarrow P \leftarrow P \cdot i$, $P=1 \cdot 3=3$ • $i \leftarrow i + 2$, $i=3+2=5$
 Pas 3: $i \leq n$? $5 \leq 8$ da $\Rightarrow P \leftarrow P \cdot i$, $P=3 \cdot 5=15$ • $i \leftarrow i + 2$, $i=5+2=7$
 Pas 4: $i \leq n$? $7 \leq 8$ da $\Rightarrow P \leftarrow P \cdot i$, $P=15 \cdot 7=105$ • $i \leftarrow i + 2$, $i=7+2=9$
 Pas 5: $i \leq n$? $9 \leq 8$ nu $\Rightarrow$ iese din ciclu

După ieșirea din ciclu, afișează ultima valoare a lui P , adică $P=105$.

Încercați singuri! - Tema nr. T.8.

Scrieți câte un algoritm, care, pentru o valoare a lui n citită de la tastatură, rezolvă fiecare dintre cerințele de mai jos:

► 1. Calculează suma numerelor naturale impare mai mici sau egale decât n . *Exemplu:* Pentru $n=9 \Rightarrow S=1+3+5+7+9$.

► 2. Afișează numerele naturale divizibile cu 3 mai mici sau egale decât n . *Exemplu:* Pentru $n=7$ se vor afișa numerele 3 și 6.

► 3. Calculează suma primelor n numere naturale impare, $S=1+3+5+\dots+(2n-1)$ (atenție, nu este același lucru cu suma numerelor naturale impare mai mici sau egale decât n cerută la problema 1!) *Exemplu:* Pentru $n=9 \Rightarrow S=1+3+5+7+9+11+13+15+17$.

► 4. Calculează valoarea expresiei $E=3-7+11+\dots+(-1)^{n+1} \cdot (4n-1)$.
Exemplu: Pentru $n=5 \Rightarrow E=3-7+11-15+19$.

Șiruri de numere citite pe rând

Problemele următoare realizează prelucrări ale unor șiruri de numere întregi sau reale.

Prin șir de numere înțelegem o succesiune de numere care respectă o anumită ordine. Numerele care alcătuiesc șirul se vor numi *elementele șirului*.

Întrucât încă nu ați învățat structurile de date care permit memorarea simultană a tuturor elementelor unui șir, ne vom limita deocamdată la o modalitate mai "primitivă" dar eficientă de prelucrare a unui șir: elementele șirului se citesc pe rând, în aceeași variabilă, până la întâlnirea unei anumite valori sau condiții care marchează sfârșitul șirului. Cei mai uzuali algoritmi sunt aceia în care se operează asupra unui șir de numere a cărui citire se încheie cu valoarea zero (care nu face parte din șir). Există două modalități de citire a unui astfel de șir:

- numerele se citesc pe rând în aceeași variabilă atâta timp cât nu s-a introdus valoarea 0;
- numerele se citesc pe rând în aceeași variabilă până la introducerea valorii 0.

Algoritm ► Numere citite pe rând cât timp nu s-a întâlnit valoarea 0

Numerele se citesc pe rând, în aceeași variabilă x , primul separat și celelalte într-un ciclu "cât timp". În ciclu: se prelucrează primul număr x (citit înaintea ciclului), apoi se citește următorul număr tot în variabila x și se revine la începutul ciclului; dacă noul număr x este diferit de zero se prelucrează, apoi se citește următorul număr x , ș.a.m.d., atâta timp cât x nu este zero.

Primul număr se citește separat, ca să aibă ce valoare să prelucreze la primul pas al ciclului. Când întâlnește $x=0$ iese din ciclu și nu ajunge să prelucreze valoarea zero (care

nu face parte din șir).

```
citește x;  
cât timp (x≠0) execută  
  început  
    <prelucrează x>  
  citește x;  
  sfârșit
```

Exemplu: Fie șirul -2,3,0; citește $x=-2$
Pas 1: $x \neq 0 ? -2 \neq 0$ da $\Rightarrow$
 • prelucrează $x=-2$, citește $x=3$
Pas 2: $x \neq 0 ? 3 \neq 0$ da $\Rightarrow$
 • prelucrează $x=3$, citește $x=0$
Pas 3: $x \neq 0 ? 0 \neq 0$ nu $\Rightarrow$ STOP

În cazul în care primul număr introdus (cel citit înaintea ciclului) este 0, condiția din linia "cât timp" este falsă încă de la prima testare, deci ciclul nu se va executa niciodată.

Aplicație R.II.9. Numărare în șir de numere a cărui citire se termină cu 0

Se dă un șir de numere care se citesc pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din șir). Să se afișeze câte numere pozitive sunt în șir.

Rezolvare

 V-ați dat seama desigur că este vorba tocmai despre algoritmul de citire a unor numere pe rând, în aceeași variabilă x , atâta timp cât nu s-a introdus valoarea zero. Tot ceea ce mai trebuie este să stabilim în ce constă prelucrarea fiecărui x citit, la fiecare pas al ciclului.

Problema cere să se numere câte dintre elementele citite sunt pozitive. Pentru aceasta vom folosi o variabilă nr . Cum anume realizăm efectiv numărarea folosind algoritmul prezentat mai sus?

- ♦ Inițializăm nr cu 0.
- ♦ Citim separat primul element al șirului, în variabila x ;
- ♦ Într-un ciclu, cât timp x este diferit de zero:
 - ✓ testăm dacă elementul x tocmai citit este pozitiv. În caz afirmativ, mărim cu 1 valoarea lui nr , prin atribuirea $nr \leftarrow nr+1$ (marcând astfel faptul că am întâlnit încă un element pozitiv în plus față de câte erau până în acel moment). Aceasta este acțiunea pe care am denumit-o generic "prelucrează x ";
 - ✓ citim următorul element al șirului, tot în variabila x .

Pseudocodul complet este următorul:

```
citește x;  
nr←0;  
cât timp (x≠0) execută  
  început  
    dacă (x>0) atunci  
      nr←nr+1;  
    citește x;  
  sfârșit  
scrie nr;
```

Încercați singuri! - Tema nr. T.9.

►1. Se citesc de la tastatură pe rând numerele unui șir, atâta timp cât nu s-a introdus valoarea 0 (aceasta nu face parte din șir). Să se calculeze suma elementelor pozitive și suma elementelor negative din șir.

Exemplu: pentru șirul $(-2, 8, 25, 7, -3, 9, 0)$, se vor afișa valorile 49 și -5.

►2. Se citesc de la tastatură n numere naturale. Realizați un algoritm care calculează media aritmetică a elementelor impare din șir.

Exemplu: pentru șirul $(-2, 8, 25, 7, -3, 9, 0)$, rezultatul cerut este $(25+7+9+(-3))/4$, adică 9.25.

Aplicație R. II.10. Maxim într-un șir de numere a cărui citire se termină cu 0

Se dă un șir de numere care se citesc pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din șir). Să se afișeze maximul dintre elementele șirului.

Rezolvare

Citirea numerelor pe rând, în aceeași variabilă x , cât timp nu s-a introdus valoarea zero, are loc după același algoritm pe care l-am prezentat înainte: citim separat primul element al șirului în variabila x ; apoi, într-un ciclu, cât timp x este diferit de 0, prelucrăm x -ul curent și citim următorul element tot în x . Folosim o variabilă max , în care actualizăm permanent maximul.

În detaliu, algoritmul se desfășoară astfel:

- ♦ Citim separat primul element al șirului, în variabila x ;
- ♦ Inițializăm maximul max cu primul element x citit înainte;
- ♦ Într-un ciclu, cât timp x este diferit de zero:
 - ✓ dacă elementul memorat în variabila x este mai mare decât maximul curent ($x > max$), atunci respectivul element devine noul maxim, fapt care se marchează prin atribuirea $max \leftarrow x$;
 - ✓ citim următorul element al șirului, tot în variabila x .

Pseudocodul complet este următorul:

```

citește x;
max ← x;
cât timp (x ≠ 0) execută
    început
        dacă (x > max) atunci max ← x
        citește x;
    sfârșit
scrie max
    
```

Exemplu: Fie șirul $-2, 3, 7, 1, 0$; citește $x = -2$; $max \leftarrow x$, $max = -2$;

Pas 1: $x \neq 0 ? -2 \neq 0$ da $\Rightarrow$

• $x > max ? -2 > -2$ nu; • citește $x = 3$

Pas 2: $x \neq 0 ? 3 \neq 0$ da $\Rightarrow$

• $x > max ? 3 > -2$ da $\Rightarrow max \leftarrow x$, $max = 3$; • citește $x = 7$

Pas 3: $x \neq 0 ? 7 \neq 0$ da $\Rightarrow$

• $x > max ? 7 > 3$ da $\Rightarrow max \leftarrow x$, $max = 7$; • citește $x = 1$

Pas 4: $x \neq 0 ? 1 \neq 0$ da $\Rightarrow$

• $x > max ? 1 > 7$ nu; • citește $x = 0$

Pas 4: $x \neq 0 ? 0 \neq 0$ nu $\Rightarrow$ STOP

Observați că, actualizând permanent variabila max în maniera descrisă, în final valoarea acestei variabile va fi cel mai mare element din tot șirul, adică 7. Algoritmul efectuează o operație inutilă: întotdeauna, la primul pas se va compara maximul cu el însuși, deci nu există posibilitatea ca valoarea variabilei max să se modifice. Am preferat acest mic inconvenient în favoarea unui algoritm foarte didactic și cu un caracter mare de generalitate.

Încercați singuri! - Tema nr. T.10.

►1. Modificați algoritmul din problema rezolvată anterioră, astfel încât el să afișeze minimul dintre elementele pare ale șirului.

Exemplu: pentru șirul $(8, -2, 25, 7, -3, 9, 0)$, se va tipări valoarea -2.

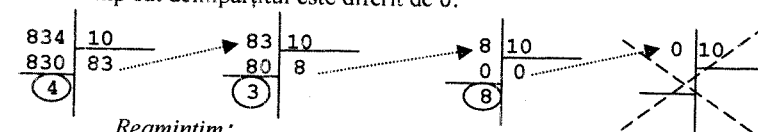
►2. Se dă un șir de numere citite pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din șir). Proiectați pseudocodul unui algoritm care determină produsul pătratelor perfecte din șir. Reamintim: un număr x este pătrat perfect dacă este egal cu "parte întreagă din radical din x ".

Exemplu: pentru șirul $(45, 25, 147, 9, 111, 1231, 0)$, se va afișa valoarea 225, reprezentând produsul pătratelor perfecte 25 și 9.

Prelucrarea cifrelor unui număr natural

Algoritm ► Extragerea cifrelor unui număr natural

Cifrele unui număr pot fi extrase de la dreapta la stânga, cu un algoritm bazat pe un ciclu de împărțiri la 10. Inițial deîmpărțitul este chiar numărul. Se împarte deîmpărțitul la 10 și restul este ultima cifră a numărului, iar câtul devine deîmpărțit pentru următoarea împărțire. Se împarte noul deîmpărțit la 10, restul este penultima cifră, câtul devine din nou deîmpărțit, se împarte noul deîmpărțit la 10, ș.a.m.d.. Procesul se repetă într-un ciclu, atâta timp cât deîmpărțitul este diferit de 0.



Reamintim:

câtul împărțirii întregi a lui x la y este x / y ;

restul împărțirii întregi a lui x la y se notează $x \% y$;

Prezentăm în continuare o variantă în care extragerea cifrelor se face într-o structură repetitivă cu test inițial.

Folosim două variabile, c și d : în c reținem fiecare cifră extrasă, iar în d actualizăm deîmpărțitul la fiecare împărțire. Cum spuneam, inițial deîmpărțitul este chiar numărul x , deci se face atribuirea $d \leftarrow x$. Într-un ciclu, cât timp deîmpărțitul d este diferit de 0, la fiecare pas:

- ♥ Extragem o cifră c , care este tocmai restul împărțirii deîmpărțitului la 10 ($c \leftarrow d \% 10$).
- ♥ Prelucrăm cifra extrasă c ;
- ♥ Câtul împărțirii tocmai efectuate, adică $[d/10]$, va deveni noul deîmpărțit d pentru împărțirea următoare ($d \leftarrow [d/10]$).

```

citește x;
d ← x;
cât timp (d ≠ 0) execută
    început
        c ← d % 10;
        <prelucrează c>
        d ← d / 10;
    sfârșit

```

Exemplu: Să descriem ciclul pentru $x=179$. Inițial, $s=0$; $d=x$, $d=179$.

- Pasul 1:* $d \neq 0$? $179 \neq 0$? *da* $\Rightarrow$ execută corpul ciclului
 $c \leftarrow d \% 10$, $c = 179 \% 10$, $c = 9$; <prelucrează cifra 9>;
 $d \leftarrow [d/10]$, $d = [179/10]$, $d = 17$
- Pasul 2:* $d \neq 0$? $17 \neq 0$? *da* $\Rightarrow$ execută corpul ciclului
 $c \leftarrow d \% 10$, $c = 17 \% 10$, $c = 7$; <prelucrează cifra 7>;
 $d \leftarrow [d/10]$, $d = [17/10]$, $d = 1$
- Pasul 3:* $d \neq 0$? $1 \neq 0$? *da* $\Rightarrow$ execută corpul ciclului
 $c \leftarrow d \% 10$, $c = 1 \% 10$, $c = 1$; <prelucrează cifra 1>;
 $d \leftarrow [d/10]$, $d = [1/10]$, $d = 0$
- Pasul 4:* $d \neq 0$? $0 \neq 0$? *nu* $\Rightarrow$ abandonează ciclul.

Aplicație R.II.11. Suma cifrelor unui număr

Realizați un algoritm care calculează suma cifrelor unui număr natural x citit de la tastatură.

Rezolvare

 V-ați dat desigur seama că vom folosi algoritmul de extragere a cifrelor de mai sus, bazat pe un ciclu de împărțiri ale unui deîmpărțit d la 10 (inițial deîmpărțitul este chiar numărul x). "Prelucrarea" cifrei c extrasă la fiecare pas constă în adăugarea ei la suma s a cifrelor lui x (unde s se inițializează cu 0). Mai concret:

- Citim de la tastatură numărul x , apoi facem inițializările $d \leftarrow x$ și $s \leftarrow 0$;

- Într-un ciclu, cât timp d este diferit de 0, la fiecare pas:
 - ✓ extragem o cifră c , ca fiind restul împărțirii deîmpărțitului la 10 ($c \leftarrow d \% 10$).
 - ✓ adăugăm cifra c la suma cifrelor s ($s \leftarrow s + c$);
 - ✓ câtul împărțirii tocmai efectuate, adică $d \div 10$, va deveni noul deîmpărțit d pentru împărțirea următoare ($d \leftarrow [d/10]$).

 Mai facem mențiunea că în loc de $c \leftarrow d \% 10$ și $s \leftarrow s + c$, putem scrie direct $s \leftarrow s + d \% 10$ (adică să nu mai memorăm cifra într-o variabilă c). Am preferat această variantă care pune în evidență mai bine procesul de "extragere" a cifrelor unui număr natural. Astfel vă va fi mai ușor să concepeți alți algoritmi pentru prelucrarea cifrelor unui număr.

Prezentăm în continuare pseudocodul în ambele variante.

```

citește x;
d ← x;
s ← 0;
cât timp (d ≠ 0)
    început
        c ← d % 10;
        s ← s + c;
        d ← d / 10;
    sfârșit
scrie s;

```

```

citește x;
d ← x;
s ← 0;
cât timp (d ≠ 0)
    început
        s ← s + d % 10;
        d ← d / 10;
    sfârșit
scrie s;

```

Încercați singuri! - Tema nr. T.11.

►1. Afișați câte cifre pare și câte cifre impare conține un număr natural x (cu maxim 8 cifre), citit de la tastatură.

Exemplu: dacă se citește $x=2354877$, programul va afișa numerele 3 și 4.

►2. Se citește de la tastatură un număr natural m . Să se afișeze cifra de rang k dat (rangul unei cifre este numărul său de ordine, numerotând cifrele de la dreapta la stânga).

Exemplu: pentru $m=452855$ și $k=3$, se va afișa valoarea 8.

►3. Se citește de la tastatură un număr natural m . Să se afișeze cifra maximă a numărului dat.

Exemplu: dacă se citește $x=2934985$, programul va afișa cifra 9.

Algoritm ► Formarea oglinditului unui număr natural

Pentru a determina oglinditul unui număr x , se aplică algoritmul de extragere a cifrelor descris în problema anterioară. Formăm oglinditul din "mers" într-o variabilă y . Inițializăm oglinditul y cu 0, apoi, la fiecare pas al ciclului, extragem o cifră c și actualizăm oglinditul după relația $y := y * 10 + c$.


```

d ← x;           // inițial deîmpărțitul d este numărul
y ← 0;           // inițializăm oglinditul y cu 0
cât timp (d ≠ 0) execută // ciclul se repetă cât timp deîmpărțitul este diferit de 0
    început
    c ← d % 10;   // extragem o cifra c, ca restul împărțirii deîmpărțitului la 10
    y ← y * 10 + c; // actualizăm oglinditul cu cifra extrasă
    d ← d / 10;
    sfârșit      // câtul împărțirii făcute va fi deîmpărțit la împărțirea următoare

```

Exemplu: fie numărul 831

```

y=0; d=x=831
831:10 → c=831%10=1, y=0*10+1=1, d=[831/10]=83
83:10 → c=83%10=3, y=1*10+3=13, d=[83/10]=8
8:10 → c=8%10=8, y=13*10+8=138, d=[8/10]=0
0:10 → stop

```

Am obținut oglinditul lui 831, care este 138.

Aplicație R.II.12. *Palindroame mai mici sau egale cu n*

Numim "oglinditul unui număr natural x ", numărul obținut prin citirea cifrelor lui x în ordine inversă, de la dreapta la stânga. *Exemplu:* oglinditul lui 2435 este 5342. Numim palindrom un număr natural egal cu oglinditul său (citit invers este același număr). *Exemplu:* numărul 232 este palindrom.

Se citește de la tastatură un număr natural n ($n \leq 20000$). Să se afișeze toate palindroamele mai mici sau egale decât n .

Exemplu: pentru $n=135$, se vor afișa numerele:

1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, 44, 55, 66, 77, 88, 99, 101, 111, 121, 131

Rezolvare

Pentru început citim valoarea lui n , apoi inițializăm cu 1 o variabilă i în care vom parcurge numerele naturale mai mici sau egale decât n , adică $1, 2, \dots, n$.

Cât timp $i \leq n$:

- determinăm oglinditul lui i cu algoritmul de mai sus, memorându-l în variabila y ;
- verificăm dacă numărul i este palindrom, adică dacă este egal cu oglinditul său y ; în caz afirmativ îl afișăm;
- variabila i ia ca valoare următorul număr natural, prin atribuirea $i \leftarrow i+1$.

```

citește n;
i ← 1;
cât timp (i ≤ n) execută
    început
    d ← i;
    y ← 0;
    cât timp (d ≠ 0) execută
        început

```

```

c ← d % 10;
y ← y * 10 + c;
d ← [d/10];
sfârșit
dacă (y == i) atunci
    scrie i;
i ← i + 1;
sfârșit

```

Încercați singuri! - Tema nr. T.12.

► 1. (Bacalaureat, iulie 2002, varianta 4)

Se citesc de la tastatură două numere naturale a și b , fiecare număr având cel mult trei cifre. Să se afișeze pe ecran, câte unul pe linie, toate numerele naturale situate în intervalul închis determinat de a și b care au suma cifrelor 15.

Exemplu: pentru $a=83$ și $b=125$ se vor afișa pe ecran valorile 87 și 96.

► 2. Se dă un șir de numere citite pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din șir). Să se afișeze acele elemente ale șirului care au proprietatea că suma cifrelor lor este divizibilă cu o valoare p dată, precum și numărul respectivelor elemente.

Exemplu: pentru șirul (45, 25, 147, 9, 111, 1231, 0), și $p=3$, algoritmul va tipări elementele 45, 147, 111 și 9, precum și valoarea 4 ce reprezintă numărul acestor elemente.

► 3. Se citește de la tastatură un număr natural n cuprins între 100 și 999. Dintre numerele naturale mai mici sau egale decât n , să se afișeze acelea care sunt divizibile cu suma cifrelor lor.

Exemplu: pentru $n=25$, se vor afișa numerele:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 18, 20, 21, 24

Interpretarea unui pseudocod dat

Aplicație R.II.13. *Pseudocod cu șir de numere*

(Bacalaureat iulie 2009, varianta 77)

Se consideră algoritmul alăturat, în care toate variabilele sunt de tip întreg.

a) Scrieți valoarea care se va afișa dacă se citesc, în această ordine, valorile: 2, 4, 6, 5, 7, 3, 9, 8, 0.

b) Scrieți un șir de date de intrare format din cel puțin trei valori care să determine afișarea valorii 0.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

```

citește a;
k ← 0;
cât timp (a ≠ 0) execută
    început
    citește b;
    dacă (a < b) atunci
        k ← k + 1;
    a ← b;
    sfârșit
    scrie k;

```

Rezolvare

a)

Prima valoare din șirul (2, 4, 6, 5, 7, 3, 9, 8, 0) este cea care se va citi separat în variabila a înaintea ciclului, iar celelalte se vor citi în ciclu, succesiv, în aceeași variabilă b . Să urmărim pașii ciclului:

Inițial $a=2$ și $k=0$.

Pasul 1: $a \neq 0$? $2 \neq 0$? da $\Rightarrow$ citește $b=4$

$a < b$? $2 < 4$? da $\Rightarrow k \leftarrow k+1, k=0+1, k=1$

$a \leftarrow b, a=4$

Pasul 2: $a \neq 0$? $4 \neq 0$? da $\Rightarrow$ citește $b=6$

$a < b$? $4 < 6$? da $\Rightarrow k \leftarrow k+1, k=1+1, k=2$

$a \leftarrow b, a=6$

Pasul 3: $a \neq 0$? $6 \neq 0$? da $\Rightarrow$ citește $b=5$

$a < b$? $6 < 5$? nu

$a \leftarrow b, a=5$

Pasul 4: $a \neq 0$? $5 \neq 0$? da $\Rightarrow$ citește $b=7$

$a < b$? $5 < 7$? da $\Rightarrow k \leftarrow k+1, k=2+1, k=3$

$a \leftarrow b, a=7$

Mai departe credem că putem prezenta doar schematic ceea ce se întâmplă:

La pasul 5: citește $b=3$, nu modifică pe k (pentru că " $7 < 3$ " este fals) și face $a=3$;

La pasul 6: citește $b=9$, face $k \leftarrow k+1$ respectiv $k=4$, apoi $a=9$;

La pasul 7: citește $b=8$, nu modifică pe k , iar $a=8$;

La pasul 8: citește $b=0$, nu modifică pe k , și în urma atribuirii $a \leftarrow b$ rezultă $a=0$.

Acum condiția " $a \neq 0$ " din linia "cât timp" este falsă, deci va ieși din ciclu. Valoarea finală a lui k , afișată în încheierea algoritmului, este $k=4$.

c)

Algoritmul citește primul număr în variabila a , apoi, în cadrul ciclului, la fiecare pas:

– citește un alt număr în variabila b ;

– compară numărul citit b cu valoarea lui a , iar dacă $b > a$ incrementează un contor k (inițializat cu 0); cum însă în a avem numărul citit anterior, rezultă că la fiecare pas se compară două numere consecutiv citite, iar contorul se mărește în cazul în care al doilea este mai mare decât primul;

– salvează numărul citit b în variabila (pentru că la pasul următor vom citi o altă valoare în variabila a , și din nou vom avea nevoie de "precedentul" pentru comparație).

Acum enunțul care să descrie acțiunea algoritmului este cum nu se poate mai clar. Îl formulăm în două moduri:

1. "Se citește un șir de numere atâta timp cât nu s-a introdus valoarea 0. Scrieți un algoritm care stabilește de câte ori se întâmplă să avem două elemente consecutive ale șirului în ordine crescătoare".

2. "Se citește un șir de numere atâta timp cât nu s-a introdus valoarea 0. Să se afișeze numărul perechilor de elemente ale șirului consecutiv citite, cu proprietatea că al doilea element al perechii este mai mare decât primul".

b)

Cu subpunctul c) deja rezolvat, răspunsul la subpunctul b) nu ar trebui să mai dea bătaie de cap. Pentru ca valoarea finală a lui k (cea afișată) să rămână cea inițială (adică 0), este necesar ca în șirul ales să alterneze monotonia, adică să nu existe două elemente consecutiv citite cu proprietatea că al doilea este mai mare decât primul. Un exemplu de astfel de șir este (2, 7, 5, 0).

Încercați singuri! - Tema nr. T.13.

► 1. (Bacalaureat iulie 2009, varianta 70)

Se consideră algoritmul alăturat, descris în pseudocod, în care toate variabilele memorează numere naturale.

a) Scrieți ce valori se vor afișa dacă pentru x se citește valoarea 19, iar pentru y se citește valoarea 4.

b) Scrieți o valoare care poate fi citită pentru variabila x și o valoare care poate fi citită pentru variabila y astfel încât valorile afișate în urma executării algoritmului să fie 2 1, în această ordine.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

d) Scrieți un algoritm echivalent cu cel dat care să nu folosească nici un fel de structură repetitivă.

citește x, y

$n \leftarrow 0$;

cât timp ($x > y$) execută

început

$x \leftarrow x - y$;

$n \leftarrow n + 1$;

sfârșit

scrie n, x

► 2. (Bacalaureat iulie 2009, varianta 68)

Se consideră algoritmul alăturat, descris în pseudocod.

S-a notat cu $x \% y$ restul împărțirii numărului natural x la numărul natural nenul y și cu $[z]$ partea întreagă a numărului real z .

a) Scrieți valoarea afișată dacă pentru n se citește valoarea 52381.

b) Scrieți o valoare care poate fi citită pentru variabila n astfel încât să se afișeze numărul 0.

c) Dacă variabila n primește ca valoare cel mai mare număr natural par de trei cifre, stabiliți câte atribuiri se execută în total pe parcursul rulării algoritmului.

citește n ;

$z \leftarrow 0$;

cât timp ($n > 0$) execută

început

$c \leftarrow n \% 10$;

```

n ← [n/10];
dacă (c < 5) atunci
    z ← z*10 + 2*c;
sfârșit
scrie z;

```

► 3. (Bacalaureat iulie 2009, varianta 25)

Se consideră algoritmul alăturat descris în pseudocod. S-a notat cu $y|x$ faptul că x este divizibil cu y .

- Scrieți care sunt valorile ce se vor afișa pentru $a=10$, $b=20$ și $c=6$.
- Găsiți un set de valori pentru variabilele a , b și c , astfel încât algoritmul să nu afișeze nici o valoare.
- Formulați un enunț scurt care să descrie acțiunea algoritmului.
- Scrieți o formulă care să calculeze câte numere se afișează dacă $a \leq b$.

```

citește a, b, c;
dacă (a > b) atunci
    început
        t ← a; a ← b; b ← t;
    sfârșit
cât timp (a <= b) execută
    început
        dacă (c | a) atunci
            scrie a;
        a ← a + 1;
    sfârșit

```

► 4. (Bacalaureat iulie 2009, varianta 31)

Se consideră algoritmul alăturat, descris în pseudocod. S-a notat cu $[x]$, partea întreagă a numărului real x .

- Scrieți valorile care se vor afișa pentru $a=9$.
- Scrieți numărul valorilor, din intervalul $[1, 5]$ care, citite pentru variabila a , determină, după executarea algoritmului alăturat, memorarea valorii 1 în variabila b .
- Scrieți în pseudocod un algoritm echivalent cu cel dat, în care să nu se utilizeze structuri repetitive sau subprograme recursive.

```

citește a;
k ← 0;
b ← [(a+1) * (a+2) / 2];
cât timp (b >= a) execută
    început
        b ← b - a;
        k ← k + 1;
    sfârșit
scrie b, k;

```

Aplicație

R.II.14. Pseudocod cu cifrele unui număr

(Bacalaureat iulie 2009, varianta 88)

Pentru pseudocodul dat, în care a este un număr natural și toate celelalte variabile sunt de tip întreg, se cere:

- Scrieți numărul care se va afișa dacă se citește valoarea 123456;
 - Dați un exemplu de o valoare care poate fi citită în variabila a , astfel încât structura repetitivă "cât timp" să aibă exact trei pași.
 - Scrieți o valoare cu exact cinci cifre care poate fi citită în variabila a , astfel încât numărul afișat să fie cel format din toate cifrele lui a luate în ordine inversă;
 - Câte numere cu exact două cifre ar putea constitui valori ale variabilei de intrare a , astfel încât în urma executării algoritmului să se afișeze valoarea citită de la tastatură?
 - Formulați un enunț scurt care să descrie acțiunea algoritmului dat.
- Notății: $x \% y$ = restul împărțirii lui x la y , și $[x/y]$ = câtul împărțirii lui x la y .

```

citește a;
p ← 1;
b ← 0;
cât timp (a ≠ 0) execută
    început
        c ← a % 10;
        dacă (a % 2 == 0) atunci
            b ← b + c * p;
        altfel
            b ← b * 10 + c;
        a ← a / 10;
        p ← p * 10;
    sfârșit
scrie b;

```

Rezolvare

a)

Pentru a deduce valoarea afișată în cazul în care se citește numărul $a=123456$, vom urmări pas cu pas evoluția ciclului "cât timp".

Plecăm de la valorile inițiale $a=123456$, $p=1$ și $b=0$.

Pasul 1: $a \neq 0$? da (pentru că 123456 este diferit de zero) $\Rightarrow$

$c \leftarrow a \% 10 \Rightarrow c = 123456 \% 10$, $c = 6$ (restul împărțirii la 10)

$a \% 2 == 0$? $123456 \% 2 == 0$? da $\Rightarrow b \leftarrow b + c * p$, $b = 0 + 6 * 1$, $b = 6$

$p \leftarrow p * 10$, $p = 1 * 10$, $p = 10$

$a \leftarrow [a/10]$, $a = [123456/10]$, $a = 12345$

Pasul 2: $a \neq 0$? da $\Rightarrow c = 12345 \% 10 = 5$

$12345 \% 2 == 0$? nu $\Rightarrow b = 6 * 10 + 5 = 65$

$p = 10 * 10 = 100$ și $a = [12345/10] = 1234$

Pasul 3: $a \neq 0$? da $\Rightarrow c = 1234 \% 10 = 4$

$1234 \% 2 == 0$? da $\Rightarrow b = 65 + 4 * 100 = 465$
 $p = 100 * 10 = 1000$ și $a = [1234 / 10] = 123$

Pasul 4: $a \neq 0$? da $\Rightarrow c = 123 \% 10 = 3$

$123 \% 2 == 0$? nu $\Rightarrow b = 465 * 10 + 3 = 4653$
 $p = 1000 * 10 = 10000$ și $a = [123 / 10] = 12$

Pasul 5: $a \neq 0$? da $\Rightarrow c = 12 \% 10 = 2$

$12 \% 2 == 0$? da $\Rightarrow b = 4653 + 2 * 10000 = 24653$
 $p = 10000 * 10 = 100000$ și $a = [12 / 10] = 1$

Pasul 6: $a \neq 0$? da $\Rightarrow c = 1 \% 10 = 1$

$1 \% 2 == 0$? nu $\Rightarrow b = 24653 * 10 + 1 = 246531$
 $p = 100000 * 10 = 1000000$ și $a = [1 / 10] = 0$

Pasul 7: $a \neq 0$? nu $\Rightarrow$ iese din ciclu

Așadar, dacă s-a citit valoarea $a = 123456$, algoritmul va afișa numărul 246531.

e)

Algoritmul construiește numărul format din cifrele pare a numărului a luate de la stânga la dreapta, urmate de cifrele impare ale lui a luate invers, de la dreapta la stânga.

Am răspuns mai întâi subpunctului e), pentru că odată ce v-ați dat seama care este acțiunea algoritmului, vă va fi mult mai ușor să înțelegeți rezolvarea celorlalte subpuncte.

b)

Observăm că:

– la începutul structurii repetitive "cât timp" se extrage în variabila c o cifră a numărului a , ca fiind restul împărțirii lui a la 10 ($c \leftarrow a \% 10$);

– la sfârșitul aceleiași structuri, în variabila a se actualizează câtul împărțirii "vechiului a " la 10; pentru a pregăti extragerea cifrei următoare ($a \leftarrow a / 10$).

Prin urmare, ciclul "cât timp" va avea atâția pași câte cifre are numărul a . Deci, pentru a se executa trei pași, putem citi în variabila a orice număr de trei cifre.

c)

Așa cum a rezultat din deducerea de la primul subpunct, doar cifrele impare ale numărului a sunt luate în ordine inversă, în vreme ce cele pare sunt abordate în ordine naturală. În consecință, pentru a obține la final un număr care să conțină doar cifre de-ale lui a luate în ordine inversă, este necesar ca numărul a să fie alcătuit numai din cifre impare. Orice valoare impară cu cinci cifre constituie un răspuns corect al acestei cerințe.

d)

Așa cum am subliniat deja, algoritmul extrage cifrele pare ale lui a în ordine naturală, de la stânga la dreapta. Deci, pentru a se obține la ieșire exact valoarea numărului memorat în a , este necesar ca acesta să conțină numai cifre pare. În continuare, nu mai trebuie altceva decât să aflăm câte numere de două cifre care conțin doar cifre pare există. Știind că numerele de două cifre sunt cele cuprinse între 10 și 99 inclusiv, împărțim domeniul lor în zece sub-intervale: [10, 19], [20, 29], ..., [90, 99]. Dintre acestea, pică din start numerele cuprinse în subintervalele [10, 19], [30, 39], [50, 59], [70, 79] și [90, 99], deoarece prima lor cifră este impară. Au rămas cinci subintervale, cele cu prima cifră pară, din care trebuie alese doar numerele care au și a doua cifră pară. Acestea sunt în mod evident valorile pare din respectivele sub-intervale, în număr de cinci pentru

fiecare sub-interval. De exemplu, în subintervalul [20, 29] avem cinci valori care conțin ambele cifre pare, și anume 20, 22, 24, 26 și 28. Cum avem cinci sub-intervale ori cinci numere fiecare, rezultă că în total sunt 25 de valori care îndeplinesc cerința acestui subpunct.

Încercați singuri! - Tema nr. T.14.

► 1. (Bacalaureat iulie 2009, varianta 45)

Se consideră algoritmul alăturat, descris în pseudocod. S-a notat cu $x \% z$ restul împărțirii numărului natural x la numărul natural nenul z .

a) Scrieți valoarea care va fi afișată de algoritm dacă se citesc, în această ordine, valorile 12 și 3.

b) Dacă valoarea citită pentru variabila y este 19, scrieți care este cea mai mare valoare care trebuie citită pentru x , astfel încât algoritmul să afișeze numărul 0.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

citește x, y ;

$z \leftarrow 1$;

$t \leftarrow 0$;

cât timp ($x >= z$) execută

început

dacă ($x \% z == y$) atunci

$t \leftarrow z$;

$z \leftarrow z + 1$;

sfârșit

scrie t ;

► 2. (Bacalaureat iulie 2009, varianta 79)

Se consideră algoritmul alăturat, reprezentat în pseudocod. S-a notat cu $x | y$ relația " x divide pe y " sau " y este divizibil cu x " și cu $[z]$ partea întreagă a numărului real z .

a) Scrieți valoarea care se va afișa dacă se citește numărul 245.

b) Scrieți o valoare care poate fi citită pentru variabila a astfel încât să se afișeze o valoare egală cu cea citită.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

citește a ;

$x \leftarrow 2$;

$k \leftarrow 0$;

cât timp ($a > 1$) execută

început

$c \leftarrow 0$;

cât timp ($x | a$) execută

început

$c \leftarrow x$;

$a \leftarrow [a / x]$;

sfârșit

```

dacă (c≠0) atunci
    k←k+x;
    x←x+1;
sfârșit
scrie k;

```

► 3. (Bacalaureat iulie 2009, varianta 89)

Se consideră algoritmul alăturat, descris în pseudocod. S-a notat cu $x \cdot y$ restul împărțirii numărului natural x la numărul natural nenul y , iar cu $[z]$ partea întreagă a numărului real z .

- a) Scrieți numărul care se va afișa dacă se citește numărul $n=3072941$.
 b) Scrieți un număr format din exact 5 cifre, ele fiind în ordine strict crescătoare, care poate fi citit astfel încât executarea algoritmului să determine afișarea unui număr egal cu cel citit.
 c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

```

citește n
t←n;
r←0;
cât timp (t>0) execută
    inceput
    dacă ((t%10)%2==1) atunci
        r←r*10+1;
    altfel
        r←r*10+t%10;
    t←[t/10];
sfârșit
n←0;
cât timp (r>0) execută
    inceput
        n←n*10+r%10;
        r←[r/10];
    sfârșit
scrie n;

```

Ați înțeles? Probleme cu răspuns scurt

Următoarele două probleme se referă la secvența de program de mai jos, în care toate variabilele memorează numere naturale.

```

citește n;
S←0; k←1;
cât timp k≤n execută
    inceput
        S←S+k;
        k←k+2;
    sfârșit
scrie S;

```

► 1. Câte valori poate primi variabila n , astfel încât, în timpul execuției secvenței să aibă loc cu totul opt atribuirii?

► 2. (Bacalaureat iulie 2008, varianta 11)

Ce reprezintă rezultatul afișat în urma execuției secvenței?

- a) suma numerelor naturale impare mai mici decât n ;
 b) suma numerelor naturale pare mai mici decât n ;
 c) suma numerelor naturale impare mai mici sau egale cu n ;
 d) suma numerelor naturale pare mai mici sau egale cu n .

► 3. (Bacalaureat iulie 2008, varianta 34)

În secvența de program de mai jos, ce valoare ar trebui scrisă în loc de "...", astfel încât, în urma execuției, să se afișeze valoarea 165?

```

i←3; E←1;
cât timp (i<...) execută
    inceput
        dacă (i%2≠0) atunci
            E←E+i*i;
        i←i+1;
    sfârșit
scrie E;

```

- a) 8 b) 9 c) 10 d) 12

Următoarele trei probleme se referă la secvența de program de mai jos, în care toate variabilele memorează numere naturale.

```

citește x,y;
x←x%10;
y←y%10;
dacă (y<x) atunci
    inceput
        aux←...;
        y←...;
        x←...;
    sfârșit;
cât timp x<=y execută
    inceput
        scrie x*10+y;
        x←x+1;
        y←y+1;
    sfârșit;

```

► 4. Precizați ce identificatori de variabile trebuie scriși, în ordine, în locul celor trei apariții ale simbolului "...", astfel încât instrucțiunea de selecție de tipul "dacă-atunci" să realizeze interschimbarea valorilor variabilelor x și y între ele.

- a) x, aux, y b) y, aux, x
 c) y, x, aux d) x, y, aux

► 5. (Bacalaureat iulie 2009, varianta 34)

Scrieți care sunt numerele afișate, dacă de la tastatură se citesc valorile $x=248$ și $y=203$.

► 6. (Bacalaureat iulie 2009, varianta 34)

Găsiți un set de valori care ar putea fi citite în variabilele x și y , astfel încât prin execuția secvenței date să se afișeze exact cinci numere.



Algoritm ▶ Cel mai mare divizor comun a două numere
- algoritmul lui Euclid cu diferențe -

Există două variante ale algoritmului lui Euclid pentru determinarea celui mai mare divizor comun a două numere naturale: cu diferențe, respectiv cu resturi. În continuare vă vom prezenta algoritmul lui Euclid cu diferențe, urmând ca în subcapitolul "Structura repetitivă cu test final" să faceți cunoștință și cu algoritmul bazat pe resturi.

Algoritmul lui Euclid cu diferențe se bazează pe scăderi care se repetă într-un ciclu. La fiecare pas, se modifică prin aceste scăderi, fie a , fie b . Dacă $a > b$ se modifică a prin scăderea $a \leftarrow a - b$, iar dacă $a < b$ se modifică b prin scăderea $b \leftarrow b - a$. Când cele două numere a și b au devenit egale se oprește ciclul, obținându-se cel mai mare divizor comun care este a (sau b). Deci ciclul se repetă cât timp a este diferit de b .

Exemplu:

Fie $a=15$; $b=6$;

Pas 1: $a \neq b$? $15 \neq 6$? da $a > b$? $15 > 6$? da $\Rightarrow a \leftarrow a - b \Rightarrow a = 15 - 6, a = 9$;

Pas 2: $a \neq b$? $9 \neq 6$? da $a > b$? $9 > 6$? da $\Rightarrow a \leftarrow a - b \Rightarrow a = 9 - 6, a = 3$;

Pas 3: $a \neq b$? $3 \neq 6$? da $a > b$? $3 > 6$? nu $\Rightarrow b \leftarrow b - a \Rightarrow b = 6 - 3, b = 3$;

Pas 4: $a \neq b$? $3 \neq 3$? nu $\Rightarrow$ iese din ciclu

C.m.m.d.c. (a, b) = $a = b = 3$

Observație: Algoritmul prezentat mai sus nu va funcționa dacă cel puțin unul din numere este negativ sau zero. Pentru a evita aceste două cazuri, poate fi pusă condiția suplimentară " $(a > 0)$ și $(b > 0)$ ".



Aplicație R.II.15. Frație zecimală

(Bacalaureat Iulie 2009, varianta 25, enunț adaptat)

Se citește de la tastatură un număr real pozitiv x , care are cel mult două cifre în partea întreagă și cel mult șapte cifre după punctul zecimal. Scrieți un program care, utilizând un algoritm eficient din punctul de vedere al memoriei utilizate și al timpului de executare, afișează pe ecran, separate printr-un spațiu, două numere naturale al căror raport este egal cu x , și a căror diferență în modul este minimă.

Exemplu: dacă se citește $x=0.375$, se vor afișa pe ecran numerele 3 și 8 (deoarece $3/8=0.375$, iar expresia $|8-3|$ este cea mai mică dintre toate diferențele în modul dintre numitorii și numărătorii fracțiilor prin care s-ar putea scrie valoarea zecimală 0.375, celelalte fracții fiind $375/1000$, $75/200$, $15/40$, etc).



Rezolvare
Problema ne cere să scriem numărul real x ca fracție de forma p/q (unde p și q sunt numere întregi), iar dintre toate fracțiile posibile, să o găsim pe aceea cu proprietatea că diferența în modul dintre numărător și numitor este cea mai mică, adică valoarea expresiei $|q-p|$, este minimă.

Pentru a scrie un număr real x ca fracție, trebuie să-l înmulțim și să-l împărțim cu o putere a lui 10, aleasă astfel încât să dispară virgula zecimală. De exemplu, $0.375=375/1000$, $2.15=215/100$, etc.). Ca să găsim cea putere a lui 10, vom înmulți pe x în mod repetat cu 10, atâta timp cât valoarea lui x mai conține încă virgula zecimală, adică atâta timp cât x nu este un număr întreg. Știind că orice număr întreg coincide cu partea întreagă din acel număr, fapt care nu este valabil în cazul numerelor cu virgulă zecimală, (de exemplu $2=[2]$ iar $2.25 \neq [2.25]$), putem trage concluzia că ciclul necesar se va executa cât timp $x \neq [x]$. În variabila q , inițializată cu 1, vom memora puterile succesive ale lui 10 prin care vom trece în timpul acestor înmulțiri repetate.

Sintetizând cele spuse, vom proiecta un ciclu în care, atâta timp cât $x \neq [x]$, la fiecare pas vom face două operații:

- ✓ înmulțim pe x cu 10, memorând rezultatul tot în x { $x \leftarrow x * 10$;};
- ✓ înmulțim pe q cu 10, pentru a trece la următoarea putere a lui 10 { $q \leftarrow q * 10$;};

În final, valoarea lui x va reprezenta numărătorul fracției, pe care îl memorăm prin atribuire în variabila p { $p \leftarrow x$;}, iar valoarea lui q va fi tocmai numitorul fracției. Într-adevăr, dacă urmărim pașii algoritmului pe numărul $x=0.375$ din exemplu, putem vedea că așa stau lucrurile: inițial $q=1$, la pasul 1 avem $x=0.375*10=3.75$ și $q=1*10=10$, apoi la pasul al doilea $x=3.75*10=37.5$ și $q=10*10=100$, și, în sfârșit, la ultimul pas $x=37.5*10=375$ iar $q=100*10=1000$. După atribuirea $p \leftarrow x$, va rezulta fracția $p/q=375/1000$.

În continuare trebuie însă să ținem cont de faptul că orice fracție de forma p/q ar putea admite simplificări succesive, în cazul în care numărătorul și numitorul au divizori comuni. Dintre toate fracțiile echivalente obținute, una singură verifică proprietatea că diferența în modul dintre numitor și numărător este cea mai mică, și anume cea care asigură simplificarea la maximum. Revenind la exemplul din enunț, fracția $375/1000$ are fracțiile echivalente, $15/40$ și $3/8$, rezultate după două simplificări. Dintre acestea, ultima, cea care asigură simplificarea la maximum, verifică proprietatea cerută ($|40-15|=25$), iar $|8-3|=5$.

Dar, ca să simplificăm la maximum o fracție, nu este necesar să implementăm toate simplificările succesive într-un ciclu ! Știm din matematică faptul că o fracție de forma p/q se simplifică la maximum prin cel mai mare divizor comun al numărătorului și numitorului. Ca atare, este suficient să aflăm c.m.m.d.c. al numerelor p și q , pe care îl vom memora în variabila d . Pentru aceasta, folosim algoritmul lui Euclid cu diferențe, prezentat anterior, dar nu îl aplicăm direct numerelor p și q , ci unor copii ale acestora, a respectiv b (pe care le obținem prin simplele atribuiri $a \leftarrow p$ și $b \leftarrow q$). De ce sunt necesare aceste copii ? Pentru că algoritmul lui Euclid modifică numerele asupra cărora se aplică, iar noi avem nevoie de valorile lui p și q nealterate, pentru a putea face după aceea simplificarea fracției !

☞ Odată determinat $d = \text{c.m.m.d.c.}(p, q)$, simplificăm fracția p/q direct prin d , cu atribuiri $p \leftarrow p/d$ și $q \leftarrow q/d$, după care, în final afișăm numărătorul p și numitorul q al fracției simplificate, care reprezintă tocmai numerele cerute.

```

citește x;
q ← 1;
cât timp (x#[x]) execută
    început
        q ← q*10;
        x ← x*10;
    sfârșit
p ← x;
scrie p, q;
a ← p;
b ← q;
cât timp (a#b) execută
    dacă (a>b) atunci
        a ← a-b
    altfel
        b ← b-a;
d ← a;
p ← p/d;
q ← q/d;
scrie p, q;

```

☞ Încercați singuri! - Tema nr. T.15.

►1. Metoda prezentată în problema rezolvată anterioară este o variantă a algoritmului lui Euclid. Algoritmul lui Euclid "clasic" pentru determinarea lui c.m.m.d.c. a două numere naturale a și b se bazează pe o succesiune de împărțiri. Inițial deîmpărțitul este a iar împărțitorul este b . La fiecare împărțire se obține un rest r , apoi, în mod repetat, împărțitorul devine deîmpărțit pentru împărțirea următoare și se împarte la restul r . Procesul se reia până când obținem restul 0, iar ultimul rest diferit de zero este cel mai mare divizor comun al lui a și b .

Exemplu:

Fie $a=48$ și $b=14$

$48:14 \Rightarrow$ restul 6; $14:6 \Rightarrow$ restul 2; $6:2 \Rightarrow$ restul 0

$\text{c.m.m.d.c.}(a, b) = 2$

Scrieți un pseudocod pentru determinarea celui mai mare divizor comun a două numere naturale date a și b , folosind algoritmul de mai sus.

►2. Se dau două numere naturale a și b reprezentând numărătorul respectiv numitorul unei fracții. Să se afișeze fracția transformată în fracție ireductibilă. O fracție devine ireductibilă simplificând-o prin cel mai mare divizor comun al numărătorului și numitorului, dacă acesta este diferit de 1.

►3. Două numere naturale se numesc prime între ele dacă cel mai mare divizor comun al lor este 1. Pentru două numere naturale n și q citite de la tastatură, să se afișeze numerele naturale mai mici sau egale decât n care sunt prime cu q .

Exemplu: pentru $n=18$ și $q=3$ se vor afișa numerele 1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17.

Aplicație R.11.16. Număr prim

Se citește de la tastatură un număr natural x diferit de 0 și 1. Scrieți un algoritm care verifică dacă numărul citit este prim sau nu, afișându-se un mesaj corespunzător.

Rezolvare

☞ Un număr natural x este prim, dacă nu are nici un divizor propriu (adică nici un divizor diferit de 1 și el însuși (valorile 1 și x nu intră în discuție, deoarece orice număr natural x este divizibil cu 1 și cu el însuși). Posibili divizori proprii ai lui x se caută printre numerele naturale până la jumătatea lui x . De ce? De exemplu, pentru $x=30$, numerele mai mari decât 15 nu au cum să fie divizori ai lui 30. Pe de altă parte, trebuie ținut cont că jumătatea unui număr impar nu este un întreg ($31:2=16.5$), deci când spunem "jumătatea lui x " ne referim la împărțirea întreagă a lui x la 2, adică $[x/2]$ (reamintim că atunci când deîmpărțitul și împărțitorul sunt numere întregi, operatorul "/" furnizează câtul întreg al împărțirii, fără partea zecimală). În concluzie, numerele printre care vom căuta divizorii lui x sunt $2, 3, \dots, [x/2]$.

Deoarece trebuie stabilit dacă este adevărată (îndeplinită) o anumită condiție, avem de-a face cu o testare logică. Este necesară o variabilă cu sens logic, adică o variabilă care poate avea numai două valori, și anume 1 și 0, corespunzătoare celor două "stări" logice posibile, ADEVĂRAT respectiv FALS. Valoarea finală a acestei variabile va indica "starea" condiției testate: 1 dacă numărul x este prim (nu are divizori), respectiv 0 în caz contrar. Denumim această variabilă ok .

☞ Procedăm prin metoda reducerii la absurd.

Presupunem inițial că numărul x nu are divizori, dând variabilei ok valoarea 1;

Folosim un ciclu cu test inițial, în care contorul i va parcurge posibii divizori ai lui x ; teoretic, aceștia sunt $2, 3, \dots, x-1$; practic însă, este posibil ca prin variabila i să nu treacă toate aceste valori, deoarece vom proiecta ciclul de așa manieră încât, în momentul în care am găsit o valoare a lui i care este efectiv divizor al lui x , să se iasă din ciclu, abandonându-se căutarea (deoarece tragem concluzia că numărul x , având cel puțin un divizor cert "în persoana" celui i , nu mai poate fi prim). Cum realizăm un astfel de ciclu?

Inițializăm i cu valoarea 2 (primul posibil divizor). Cât timp variabila i nu a parcurs încă toți posibii divizori ($i \leq x/2$) ȘI nu am găsit încă nici un divizor al lui x ($ok=1$):

- ✓ testăm dacă valoarea curentă a lui i este divizor al lui x , adică dacă restul împărțirii lui x la i este 0 ($x \% i == 0$). În caz afirmativ, numărul x nu mai poate fi prim (am găsit un divizor cert al său, "în persoana" respectivului i), și vom marca acest fapt dând variabilei ok valoarea 0.
- ✓ trecem la următorul posibil divizor, prin incrementarea lui i ($i \leftarrow i+1$).

```

ok←1;
i←2;
cât timp (i<= x / 2) ȘI (ok==1) execută
    început
        dacă (x % i==0) execută
            ok←0;
            i←i+1;
        sfârșit

```

Este momentul să înțelegi mai profund de ce condiția de continuare a ciclului este "cât timp (i<=x/2) ȘI (ok==1)". Să zicem că la un pas al ciclului, valoarea pe care o are i la acel pas este divizor al lui x. În acest caz, condiția "x%i==0" din linia "dacă" este îndeplinită, deci variabila ok va primi valoarea 0. Urmează incrementarea lui i care încheie pasul, apoi imediat după aceea se revine în linia "cât timp". Dar aici condiția "ok==1" nu mai este îndeplinită (variabila ok tocmai a primit valoarea 0), fapt care dictează întreruperea "forțată" a ciclului. Adică exact ceea ce am urmărit, să abandonăm ciclul în momentul în care am găsit un divizor cert al lui x ! Dar cel mai bine se va vedea cum stau lucrurile dacă urmărești pas cu pas un exemplu de funcționare.

Exemplu: Fie x=15.

Divizorii acestuia sunt 3 și 5. Pentru descoperirea lor, posibili divizori care ar trebui "trecuri" prin variabila i sunt 2, 3, ..., x / 2, deci i=2, 3, 4, 5, 6, 7.

Inițial ok←1 și i←2

Pasul 1: (i<=x/2) ȘI (ok==1) ? da ⇒
 x%i==0 ? 15%2=0 ? nu
 i←i+1, i=2+1=3

Pasul 2: (i<=x/2) ȘI (ok==1) ? da ⇒
 x%i==0 ? 15%3=0 ? da ⇒ ok←0
 i←i+1, i=3+1=4

Pasul 3: (i<= x / 2) ȘI (ok==1) ? nu ⇒ iese din ciclu

După al doilea pas, valoarea 0 a variabilei ok marchează faptul că numărul x nu mai este prim (am găsit pe i=3 ca divizor). Mai departe, se începe pasul al treilea, cu testarea condiției "(i<=x/2) ȘI (ok==1)" din linia "cât timp". Dar această condiție nu mai este îndeplinită, pentru că variabila ok tocmai a primit valoarea 0, fapt care determină încheierea ciclului !

Acum se vede clar de ce am atașat și expresia "ok==1" la condiția din linia "cât timp". Ce s-ar fi întâmplat în exemplul anterior dacă condiția ar fi fost doar "cât timp (i<=x/2)" ? La pasul al treilea nu s-ar mai fi încheiat ciclul, ci s-ar fi testat mai departe în mod inutil toți ceilalți posibili divizori rămași, adică ciclul s-ar fi executat și pentru i=4, 5, 6, 7 ! Pierdere de timp, în contextul în care concluzia că numărul x=15 nu este prim a rezultat fără echivoc după pasul al doilea !

Dacă în schimb x ar fi avut valoarea 13, atunci condiția "x%i=0" nu ar fi fost îndeplinită niciodată, deci variabila ok ar fi rămas cu valoarea inițială 1 (ciclul s-ar fi parcurs până la capăt, fără să se producă vreo ieșire "forțată").

În final se testează valoarea variabilei ok, și în funcție de aceasta se afișează un mesaj din care să reiese dacă numărul x este prim sau nu.

Prezentăm în continuare pseudocodul complet.

```

citește x;
ok←1;
i←2;
cât timp (i<=x/2 ȘI ok==1) execută
    început
        dacă (x%i==0)
            ok←0;
            i←i+1;
        sfârșit
    dacă (ok==1) execută
        scrie "Numărul este prim";
    altfel
        scrie "Numărul NU este prim";

```

Chiar și așa, algoritmul nu este 100 % eficient, deoarece tot mai pot apărea operații inutile. Să încercăm o îmbunătățire a lui, în sensul eliminării tuturor situațiilor "redundante". În primul rând, trebuie constatat faptul că numerele pare nu sunt prime, deoarece au un divizor cert, "în persoana" lui 2. Excepție face numărul 2, care, chiar dacă este par, este considerat număr prim. În consecință vom introduce niște testări suplimentare, caz în care pseudocodul ar arăta schematic astfel:

```

citește x;
dacă (x%2==0) atunci
    dacă (x==2) atunci
        ok←1;
    altfel
        ok←0;
    altfel

```

<aplică algoritmul de căutare a divizorilor>

Așadar se caută divizorii doar pentru numerele x impare. Dar numerele impare nu pot avea divizori pari, deci ar fi util ca, dintre posibili divizori ai lui x să-i parcurgem în variabila i doar pe cei impari. Cum putem face acest lucru ? Inițializăm înaintea ciclului pe i cu valoarea 3 (primul divizor impar), iar în ciclu, la sfârșitul fiecărui pas trecem la următorul divizor impar prin atribuirea i←i+2 !

Pseudocodul "îmbunătățit" arată astfel:

```

citește x;
dacă (x%2==0) atunci
    dacă (x==2) atunci
        ok←1;
    altfel
        ok←0;
    altfel
    început
        i←3;
        cât timp (i<=x/2 ȘI ok==1) execută
            început
                dacă (x%i==0) atunci
                    ok←0;
                    i←i+1;
            sfârșit

```



```
sfârșit
dacă (ok==1) atunci scrie "Numărul este prim";
altfel scrie "Numărul NU este prim";
```

Încercați singuri! - Tema nr. T.16.

► 1. Se citește de la tastatură un număr natural n . Să se afișeze toate numerele naturale prime mai mici sau egale cu n .

Exemplu: pentru $n=60$, se vor tipări numerele:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 51, 53, 59.

► 2. Se dă un șir de numere citite pe rând de la tastatură, care se termină cu valoarea 0. Să se afișeze câte elemente numere prime conține șirul citit.

Exemplu: pentru șirul (26, 51, 13, 125, 59, 0), programul va tipări valoarea 3, deoarece în șir sunt trei numere prime, în speță 51, 13 și 59.



II.4.2. Structura repetitivă cu test final

Pentru ilustrarea acestui tip de ciclu, vom folosi tot exemplul dat la ciclul cu test inițial.

Exemplu:

Se citește de la tastatură trei numere întregi. Să se afișeze suma lor.

Reamintim algoritmul pe scurt:

Variabila i , inițializată cu 0, numără câți pași are ciclul. La fiecare pas se execută secvența de instrucțiuni {citește x ; $S \leftarrow S+x$; } și se incrementează valoarea lui i cu 1. Ciclul se execută cât timp $i \leq 3$. În final se afișează ultimul S .

```
S ← 0;
i ← 1;
cât timp (i ≤ 3) execută
    început
        citește x;
        S ← S+x;
        i ← i+1;
    sfârșit
scrie S;
```

Așa cum deja știți, structura repetitivă folosită în exemplul de mai sus se numește *structură repetitivă cu test inițial* (sau *ciclu cu test inițial*), deoarece testarea condiției are loc la început. În limbajul C++ putem însă implementa și o așa-numită *structură repetitivă cu test final*, caracterizată prin faptul că testarea condiției are loc la sfârșit.

O astfel de structură se mai numește și ciclu "execută-cât timp".

Iată cum arată pseudocodul exemplului nostru "transcris" în varianta cu structură repetitivă cu test final:

```
S ← 0;
i ← 1;
execută
    început
        citește x;
        S ← S+x;
        i ← i+1;
    sfârșit
cât timp (i ≤ 3);
scrie S;
```

Să urmărim execuția ciclului pas cu pas, dacă de la tastatură se introduc în ordine numerele 11, 12, 13:

Inițial $S \leftarrow 0$; $i \leftarrow 1$;

Pas 1: • citește $x=11$; • $S \leftarrow S+x$, $S=0+11=11$; • $i \leftarrow i+1$, $i=1+1=2$; • $i \leq 3$? $2 \leq 3$? da

Pas 2: • citește $x=12$; • $S \leftarrow S+x$, $S=11+12=23$; • $i \leftarrow i+1$, $i=2+1=3$; • $i \leq 3$? $3 \leq 3$? da

Pas 3: • citește $x=13$; • $S \leftarrow S+x$, $S=23+13=36$; • $i \leftarrow i+1$, $i=3+1=4$; • $i \leq 3$? $4 \leq 3$? nu

⇒ iese din ciclu.

Prezentăm și varianta cu ciclu "execută-cât timp" a problemei dată ca generalizare la cicluri cu test inițial.

Să se calculeze suma a n numere naturale citite pe rând de la tastatură (unde n este un număr natural nenul, cunoscut).

Modificările sunt minore și mult prea evidente pentru a mai necesita alte comentarii: în loc de 3 vom avea n numere, unde n se citește de la tastatură înaintea ciclului.

```
citește n;
S ← 0; i ← 1;
execută
    început
        citește x;
        S ← S+x;
        i ← i+1;
    sfârșit
cât timp (i ≤ n);
scrie S;
```

Sintaxă: generală a ciclului cu test final:

<pre>execută <secv> cât timp (<expr>);</pre>	– unde $\langle \text{expr} \rangle$ este o condiție (expresie logică), iar $\langle \text{secv} \rangle$ este o secvență formată dintr-una sau mai multe instrucțiuni, numită <i>corpul ciclului</i>.
--	---

Principiul de funcționare:

Repetă execuția secvenței de instrucțiuni $\langle \text{secv} \rangle$, până când $\langle \text{expr} \rangle$ ia valoarea de adevăr "ADEVĂRAT" (până când condiția devine adevărată).

1. se execută secvența de instrucțiuni **<secv>**;

2. se evaluează expresia logică **<expr>**;

- dacă condiția dată de **<expr>** este adevărată, adică dacă expresia **<expr>** are valoarea 1, atunci se revine la pasul "1.";
- în caz contrar, se iese din ciclu și se execută prima instrucțiune de după ciclu.

Observații:

- ✓ La fel ca și în cazul structurii repetitive cu test inițial, expresia **<expr>** din linia "cât timp" se va pune între paranteze rotunde (așa cum pretinde sintaxa limbajului C++);
- ✓ În cazul în care secvența de instrucțiuni **<secv>** ce reprezintă corpul ciclului conține mai mult de o instrucțiune, ea va fi cuprinsă între acoladă deschisă și închisă (la fel ca și la structura repetitivă cu test inițial);
- ✓ Deoarece testarea are loc la sfârșit, corpul unui ciclu cu test final se va executa cel puțin o dată.

Reluarea unor algoritmi care au fost prezentați și cu structuri cu test inițial

Aplicație R.II.17. Produs de numere naturale impare

Scrieți un algoritm care calculează produsul numerelor naturale impare nenule mai mici sau egale decât o valoare n dată.

Exemplu: pentru $n=8$ se va calcula $P=1*3*5*7$

Precizare: Această problemă a fost rezolvată și în varianta cu ciclu cu test inițial, în cadrul aplicației R.II.10. din lecția "II.4.1. Structura repetitivă cu test inițial".

Rezolvare

Calculăm produsul într-un ciclu, folosind o variabilă i , ale cărei valori sunt pe rând termenii produsului, 1, 3, 5, ... Inițial $i \leftarrow 1$ (prima valoare impară nenulă), $P \leftarrow 1$ și executăm un ciclu pe care îl putem descrie astfel:

execută:

- ✓ înmulțește valoarea lui i cu produsul P a numerelor impare ($P \leftarrow P * i$, adică "noul P este vechiul P înmulțit cu i "),
- ✓ variabila i ia următoarea valoare impară ($i \leftarrow i + 2$)

cât timp valoarea lui i este mai mică sau egală decât valoarea lui n .

Pentru a exista "vechiul P " la execuția primului pas al ciclului, variabila P trebuie inițializată cu o valoare înainte de ciclu. Valoarea inițială a lui P este 1, ca să nu influențeze rezultatul înmulțirii de la primul pas ($P \leftarrow P * i = 1 * 1$).

citește n ;

$P \leftarrow 1$;

$i \leftarrow 1$;

execută

Presupunem că s-a citit $n=6$. $P \leftarrow 1$; $i \leftarrow 1$;

Pas 1: $P \leftarrow P * i$, $P=1*1=1$; $i \leftarrow i+2$, $i=1+2=3$;

$i \leq n$? $3 \leq 6$ da $\Rightarrow$ reia ciclul

Pas 2: $P \leftarrow P * i$, $P=1*3=3$; $i \leftarrow i+2$, $i=3+2=5$

inceput
 $P \leftarrow P * i$;
 $i \leftarrow i + 2$;
sfârșit
cât timp ($i \leq n$)
scrie P ;

• $i \leq n$? $5 \leq 6$ da $\Rightarrow$ reia ciclul
Pas 3: $P \leftarrow P * i$, $P=3*5=15$; $i \leftarrow i+2$, $i=5+2=7$
• $i \leq n$? $7 \leq 6$ nu $\Rightarrow$ abandonează ciclul
După ciclu, afișează ultima valoare a lui P , adică $P=15$.

Încercați singuri !

Folosind structuri repetitive cu test final, scrieți algoritmi pentru toate problemele propuse în tema T.10. din lecția "2.4.1. Structura repetitivă cu test inițial" (capitolul II).

Algoritm Citirea unui șir pe rând până la introducerea valorii 0

Numerele se citesc pe rând, în aceeași variabilă x , primul separat și celelalte într-un ciclu execută-cât timp. În ciclu: se prelucrează primul număr x (cel citit înainte de ciclu), apoi se citește următorul număr tot în variabila x ; se testează și dacă nu este zero se reia ciclul, se prelucrează acest nou număr, apoi se citește următorul număr x , se testează, ș.a.m.d., până când se citește $x=0$. În acest moment, testând valoarea lui x și constatând că este zero, se abandonează ciclul, deci nu prelucrează valoarea zero. Primul x se citește separat pentru a avea ce valoare să prelucreze la primul pas al ciclului.

citește x ;
execută
inceput
prelucrează x ;
citește x ;
sfârșit
cât timp ($x \neq 0$);

Exemplu: Fie șirul -2, 3, 0; citește $x=-2$

Pas 1: prelucrează $x=-2$, citește $x=3$,
testează $x \neq 0$? $3 \neq 0$ da

Pas 2: prelucrează $x=3$, citește $x=0$,
testează $x \neq 0$? $0 \neq 0$ nu $\Rightarrow$ STOP

Aplicație R.II.18. Numărare într-un șir de numere a cărui citire se termină cu 0

Se dă un șir de numere citite pe rând de la tastatură, până la întâlnirea valorii 0 (care nu face parte din șir). Presupunând că șirul conține cel puțin un număr nenul, să se afișeze câte numere pozitive sunt în șir.

Precizare: Această problemă a fost rezolvată și în varianta cu ciclu cu test inițial, în cadrul aplicației R.II.11. din lecția "II.4.1. Structura repetitivă cu test inițial", cu deosebirea că acolo numerele se citeau pe rând cât timp nu s-a introdus valoarea 0.

Rezolvare

La fiecare pas al ciclului, în cadrul acțiunii denumită generic "prelucrează x ", se testează dacă valoarea curentă a lui x este pozitivă, iar în caz afirmativ se incrementează cu 1 valoarea variabilei nr care numără câte elemente pozitive conține șirul (nr este inițializat cu 0 înainte de ciclu). Considerăm că pseudocodul complet nu mai necesită alte comentarii.

```

citește x;
nr ← 0;
execută
  început
    dacă (x > 0) atunci
      nr ← nr + 1;
      citește x;
    sfârșit
  cât timp (x ≠ 0);
scrie nr;

```

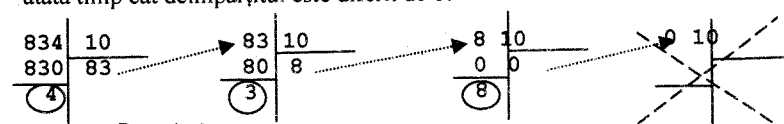
Precizarea din enunț conform căreia șirul conține cel puțin un număr nenul înlătură un caz particular în care algoritmul de mai sus nu ar funcționa. Dacă prima valoare introdusă (cea citită înaintea ciclului) este 0, atunci se intră în ciclu unde se cere o altă citire care nu există!

Încercați singuri!

Folosind structuri repetitive cu test final, scrieți algoritmi pentru toate problemele propuse în cadrul temelor T.11. și T.12., din lecția "2.4.1. Structura repetitivă cu test inițial" (capitolul II).

Algoritm ► Extragerea cifrelor unui număr natural

Cifrele unui număr pot fi extrase de la dreapta la stânga, cu un algoritm bazat pe un ciclu de împărțiri la 10. Inițial deîmpărțitul este chiar numărul. Se împarte deîmpărțitul la 10 și restul este ultima cifră a numărului, iar câtul devine deîmpărțit pentru următoarea împărțire. Se împarte noul deîmpărțit la 10, restul este penultima cifră, câtul devine din nou deîmpărțit, se împarte noul deîmpărțit la 10, ș.a.m.d.. Procesul se repetă într-un ciclu, atâta timp cât deîmpărțitul este diferit de 0.



Reamintim:

- ✓ câtul împărțirii întregi a lui x la y este x / y ;
- ✓ restul împărțirii întregi a lui x la y se notează $x \% y$;

Prezentăm în continuare o variantă în care extragerea cifrelor se face într-o structură repetitivă cu test inițial.

Folosim două variabile, c și d : în c reținem fiecare cifră extrasă, iar în d actualizăm deîmpărțitul la fiecare împărțire. Cum spuneam, inițial deîmpărțitul este chiar numărul x , deci se face atribuirea $d \leftarrow x$. Urmează ciclul propriu-zis:

Execută

- ✓ extragem o cifră c , care este tocmai restul împărțirii deîmpărțitului la 10 ($c \leftarrow d \% 10$).
- ✓ Prelucrăm cifra extrasă c ;
- ✓ câtul împărțirii tocmai efectuate, adică $[d/10]$, va deveni noul deîmpărțit d pentru împărțirea următoare ($d \leftarrow [d/10]$).

Cât timp deîmpărțitul d este diferit de 0.

```

citește x;
d ← x;
execută
  început
    c ← d % 10;
    <prelucrează c>
    d ← [d/10];
  sfârșit
  cât timp (d ≠ 0)

```

Algoritm ► Formarea oglinditului unui număr natural

Pentru a determina oglinditul unui număr x , se aplică algoritmul de extragere a cifrelor descris în problema anterioară. Formăm oglinditul din "mers" într-o variabilă y . Inițializăm oglinditul y cu 0, apoi, la fiecare pas al ciclului, extragem o cifră c și actualizăm oglinditul după relația $y := y * 10 + c$.

```

d ← x;           // inițial deîmpărțitul d este numărul
y ← 0;           // inițializăm oglinditul y cu 0
execută
  început
    c ← d % 10;   // extragem o cifră c, ca restul împărțirii deîmpărțitului la 10
    y ← y * 10 + c; // actualizăm oglinditul cu cifra extrasă
    d ← [d/10];   // câtul împărțirii făcute va fi deîmpărțit la împărțirea următoare
  sfârșit
  cât timp (d ≠ 0); // ciclul se repetă până când deîmpărțitul devine 0

```

Exemplu: fie numărul 831

$y=0; d=x=831$			
$831:10 \rightarrow$	$c=831 \% 10=1,$	$y=0*10+1=1,$	$d=[831/10]=83$
$83:10 \rightarrow$	$c=83 \% 10=3,$	$y=1*10+3=13,$	$d=[83/10]=8$
$8:10 \rightarrow$	$c=8 \% 10=8,$	$y=13*10+8=138,$	$d=[8/10]=0$
$0:10 \rightarrow$	stop		

Am obținut oglinditul lui 831, care este 138.

Aplicație R.11.19. Palindroame mai mici sau egale cu n

Numim "oglinditul unui număr natural x ", numărul obținut prin citirea cifrelor lui x în ordine inversă, de la dreapta la stânga. Exemplu: oglinditul lui 2435 este 5342. Numim palindrom un număr natural egal cu oglinditul său (citit invers este același număr). Exemplu: numărul 232 este palindrom.

Se citește de la tastatură un număr natural n ($n \leq 20000$). Să se afișeze toate palindroamele mai mici sau egale decât n .

Exemplu: pentru $n=135$, se vor afișa numerele:

1 2 3 4 5 6 7 8 9 11 22 33 44 55 66 77 88 99 101 111 121 131

Precizare: Această problemă a fost rezolvată și în varianta cu ciclu cu test inițial, în cadrul aplicației R.II.14. din lecția "II.4.1. Structura repetitivă cu test inițial".

Rezolvare

Pentru început citim valoarea lui n , cu validarea acesteia: execută citirea lui n , cât timp valoarea pe care am introdus-o este mai mică decât 0 SAU mai mare decât 20000.

Inițializăm cu 1 o variabilă i în care vom parcurge numerele naturale mai mici sau egale decât n , adică $1, 2, \dots, n$.

Execută

- determinăm oglinditul lui i cu algoritmul de mai sus, memorându-l în variabila y ;
- verificăm dacă numărul i este palindrom, adică dacă este egal cu oglinditul său y ; în caz afirmativ îl afișăm;
- variabila i ia ca valoare următorul număr natural, prin atribuirea $i \leftarrow i+1$.

Cât timp $i \leq n$.

```

execută
  citește n;
  cât timp (n<0 SAU n>20000);
  i←1;
  execută
    început
      d←i; y←0;
      execută
        început
          c←d%10; y←y*10+c; d←[d/10];
          sfârșit
        cât timp (d≠0);
        dacă (y==i)
          scrie i;
          i←i+1;
        sfârșit
      cât timp (i≤n);

```

Încercați singuri!

Folosind structuri repetitive cu test final, scrieți algoritmi pentru toate problemele propuse în cadrul temei T.14., din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Aplicație R.II.20. Interpretarea unui pseudocod dat

(Bacalaureat iulie 2009, varianta 8)

Se consideră programul pseudocod alăturat, în care toate variabilele pot primi ca valori doar numere naturale.

a) Ce valoare se va afișa pe ecran, dacă de la tastatură se citește $n=1764$?

b) Scrieți o valoare de trei cifre care poate fi citită pentru variabila n , astfel încât să se afișeze valoarea 0.

c) Dați un exemplu de valoare pe care ar putea-o primi variabila n , astfel încât atribuirea $p \leftarrow p+1$ să se execute de exact cinci ori.

d) Formulați un enunț scurt care să descrie în cuvinte acțiunea algoritmului.

e) Rescrieți algoritmul, înlocuind structura repetitivă "cât timp" cu o structură repetitivă echivalentă cu test final.

f) Scrieți un algoritm echivalent cu cel dat, care să nu folosească nici un fel de structură repetitivă.

```

citește n;
k←1;
d←2;
execută
  început
    p←0;
    cât timp (n%d==0) execută
      început
        p←p+1;
        n=[n/d];
      sfârșit
    dacă (p%2==1) atunci
      k←0;
      d←d+1;
    sfârșit
  cât timp (n>1);
  scrie k

```

Rezolvare

a)

Urmărim algoritmul pas cu pas pentru $n=1764$.

Inițial $k=1$ și $d=2$. Ciclul "execută-cât timp" se derulează astfel:

Pasul 1: $p=0$, intră în ciclul "cât timp":

Pasul 1.1.: $n \% d == 0$? $1764 \% 2 == 0$? da $\Rightarrow$

$p \leftarrow p+1, p=0+1, p=1$

$n \leftarrow [n/d], n=[1764/2], n=882$

Pasul 1.2.: $n \% d == 0$? $882 \% 2 == 0$? da $\Rightarrow$

$p \leftarrow p+1, p=1+1, p=2$

$n \leftarrow [n/d], n=[882/2], n=441$

Pasul 1.3.: $n \% d == 0$? $441 \% 2 == 0$? nu $\Rightarrow$ iese din ciclul "cât timp"

$p \% 2 == 1$? $2 \% 2 == 1$? nu

$d \leftarrow d+1, d=2+1, d=3$

$n > 1$? $441 > 1$? nu $\Rightarrow$ reia ciclul "execută-cât timp".

Pasul 2: $p=0$, intră din nou în ciclul "cât timp":

Pasul 2.1.: $441 \% 3 == 0$? da $\Rightarrow$

$p \leftarrow p+1, p=0+1, p=1$

$n \leftarrow [n/d], n=[441/3], n=147$

Pasul 2.2.: $n \bmod d = 0$? $147 \bmod 3 = 0$? da $\Rightarrow$

$p \leftarrow p+1, p=1+1, p=2$

$n \leftarrow [n/d], n=[147/3], n=49$

Pasul 2.3.: $n \bmod d = 0$? $49 \bmod 3 = 0$? nu $\Rightarrow$ iese din ciclul "cât timp"

$d \leftarrow d+1, d=3+1, d=4$

$n > 1$? $49 > 1$? da $\Rightarrow$ reia ciclul "execută-cât timp".

Urmează *pasul 3*, cu $n=49$ și $d=4$. Aici pleacă din nou de la $p=0$, dar ciclul "cât timp" nu se execută niciodată, deoarece $49 \nmid 4$. Drept consecință, nici instrucțiunea "dacă" nu are nici un efect ($0 \nmid 1$) și se execută doar atribuirea $d \leftarrow d+1$, de unde rezultă $d=5$.

La *pasul 4*, cu $d=5$, se petrece exact aceeași situație ca la *pasul 4*, rezultând în final $d=6$. Nici *pasul 5* nu este mai breaz, el conducând doar la atribuirea $d=7$. Abia acum se mai întâmplă ceva în ciclul "execută": se va executa ciclul "cât timp" de două ori, rezultând mai întâi $n=7$ și $p=1$, apoi $n=1$ și $p=2$. Nici la acest pas condiția " $p \bmod 2=0$ " din linia "dacă" nu este îndeplinită, însă condiția " $n>1$ " din linia "cât timp" a devenit falsă, fapt care produce încheierea primului ciclu.

Observăm că variabila k a rămas cu valoarea inițială 1, valoare care se va tipări în încheierea algoritmului.

d)

Așa cum am procedat până acum la toate problemele de acest gen, rezolvăm mai întâi subpunctul care ne cere să descriem acțiunea algoritmului, pentru că astfel vom vedea mai ușor soluția celorlalte subpuncte.

În variabila d trec pe rând numerele naturale începând cu 2 (adică 2, 3, 4, ...). Pentru fiecare valoare a lui d , algoritmul face două lucruri:

- numără în variabila p de câte ori se împarte n la d , adică la ce putere apare d ca factor în descompunerea lui n .

- testează dacă puterea p este impară, iar în caz afirmativ atribuie lui k valoarea 0 (unde k a fost inițializat cu 1 înaintea primului ciclu).

Prin urmare, la întâlnirea unui factor d care apare la o putere p impară în descompunerea lui n , variabila k primește valoarea 0, iar dacă toți factorii apar la puteri pare (cum a fost cazul lui $n=1764$ de la primul subpunct), atunci variabila k rămâne cu valoarea inițială 1. Cu alte cuvinte, variabila k are sens logic, jucând rolul unui "semafor", pentru a testa, prin reducere la absurd, "starea" unei condiții. Acum enunțul care să descrie acțiunea algoritmului este evident:

"Scrieți un algoritm care citește de la tastatură un număr natural n , apoi verifică dacă există cel puțin un factor care să apară la putere impară în descompunerea lui n în factori, și afișează rezultatul logic al testării".

b)

Pentru ca variabila k să primească valoarea 0, este necesar ca numărul n să conțină cel puțin un factor care să apară la putere impară în descompunere. Dacă luăm factorul $d=3$ și puterea $p=5$, atunci trebuie ales un număr care să conțină în descompunerea sa pe 3^5 , adică 243. Întrucât se cere ca numărul căutat să aibă exact trei cifre, acest 243 trebuie înmulțit cu ceva astfel încât rezultatul să nu depășească valoarea 999. O soluție ar fi $243 \cdot 4$, adică 972. Într-adevăr, descompunerea lui 972 în factori este $3^5 \cdot 2^2$, iar pentru factorul $d=3$ și puterea $p=5$ condiția " $p \bmod 2=1$ " care determină atribuirea " $k=0$ " este îndeplinită.

c)

Atribuirea $p \leftarrow p+1$ are loc în cadrul ciclului "cât timp". Pentru ca ea să se execute în total de cinci ori, putem alege, spre exemplu, cazul în care primul ciclu, cel de

tipul "execută-cât timp", are doi pași, și la primul pas atribuirea în cauză se execută de trei ori iar la al doilea pas de două ori. Altfel spus:

- la primul pas al ciclului "mare", pentru $d=2$, să obținem în final $p=3$ (adică ciclul "cât timp" din interior să aibă trei pași, pentru că variabila p pleacă de la valoarea 0);

- după al doilea pas, pentru $d=3$, să rezulte $p=2$.

Numărul n care respectă succesiunea de pași descrisă mai sus este $n=2^3 \cdot 3^2$, adică $n=72$.

Desigur, o soluție la fel de corectă ar fi fost și $n=2^2 \cdot 3^3=108$, sau $n=2^1 \cdot 3^4=162$, sau $n=2^4 \cdot 3^1=48$ (important este ca suma puterilor să fie 5).

e)

Condiția "cât timp $n \bmod d=0$ " se înlocuiește prin "execută ... cât timp $n \bmod d \neq 0$ ". Algoritmul echivalent este următorul:

```

citește n;
k ← 1;
d ← 2;
execută
    p ← 0;
    execută
        p ← p + 1;
        n ← [n/d];
        cât timp (n % d == 0);
        dacă (p % 2 == 1) atunci
            k ← 0;
        d ← d + 1;
    cât timp (n > 1);
scrie k

```

f)

Așa cum am spus, algoritmul afișează valoarea 1, dacă toți factorii din descompunerea lui n apar numai la puteri pare. Această proprietate aparține numerelor naturale care sunt pătrate perfecte (de exemplu $81=3^4$, $144=2^4 \cdot 3^2$, etc.). Dar știm din matematică faptul că un număr natural n este pătrat perfect, dacă și numai dacă "radical din n este egal cu parte întreagă din radical din n ". Prin urmare, putem concepe un algoritm echivalent, care, fără să folosească nici un fel de structură repetitivă, nu face altceva decât să testeze dacă este îndeplinită condiția de mai sus. În caz afirmativ face atribuirea $k \leftarrow 1$, iar în caz contrar avem $k \leftarrow 0$. Lăsăm ca exercițiu scrierea pseudocodului aferent acestui algoritm, pentru că este de-a dreptul banal.

Încercați singuri! - Tema nr. T.20.

► 1. (Bacalaureat iulie 2009, varianta 66)

Se consideră algoritmul alăturat, descris în pseudocod, în care s-a notat cu $a \div b$ restul împărțirii numărului întreg a la numărul întreg nenul b .

a) Scrieți ce se afișează în urma executării algoritmului dacă se citesc, în această ordine, valorile 10, 13, 46, 70, 35, 0.

b) Scrieți un șir de valori care pot fi citite pentru variabila x , astfel încât programul să afișeze, în urma executării, mesajul DA.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

d) Scrieți în pseudocod un algoritm echivalent cu cel dat, în care să se înlocuiască structura "execută...cât timp", cu o structură repetitivă cu test inițial.

```
n ← 0;
execută
  citește x;
  dacă (x ≠ 0) atunci
    dacă (x % 5 == 0) atunci
      n ← n + 1;
    altfel
      n ← n - 1;
  cât timp (x ≠ 0);
  dacă n == 0 atunci
    scrie "DA";
  altfel
    scrie "NU";
```

► 2 (Bacalaureat iulie 2009, varianta 75)

Se consideră algoritmul alăturat, descris în pseudocod, în care s-a notat cu $[a/b]$ câtul împărțirii întregi a numărului natural a la numărul natural nenul b .

a) Scrieți ce se va afișa dacă se citesc, în această ordine, valorile: 62521, 2571, 2, 56, 614.

b) Scrieți un set de date de intrare pentru care algoritmul să afișeze valoarea 12345678.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

d) Scrieți în pseudocod un algoritm echivalent cu cel dat, în care structura "execută...cât timp" să fie înlocuită cu o structură repetitivă de alt tip.

```
a ← 0;
k ← 0;
execută
  citește x;
  cât timp (x > 99) execută
    x ← [x/10];
  dacă (x > 9) atunci
    început
      a ← a * 100 + x;
      k ← k + 1;
    sfârșit
  cât timp (k ≠ 4);
scrie a;
```

► 3 (Bacalaureat iulie 2009, varianta 69)

Se consideră algoritmul alăturat, descris în pseudocod. S-a notat cu $x \% y$ restul împărțirii numărului natural x la numărul natural nenul y și cu $[z]$ partea întreagă a numărului real z .

a) Scrieți valoarea afișată dacă pentru x se citește valoarea 4589 și pentru y se citește valoarea 723.

b) Deduceți ce valori pot fi citite pentru x și y astfel încât valoarea afișată să fie 200.

```
citește x, y;
t ← 0; u ← 1;
execută
  dacă (x % 10 > y % 10) atunci
    z ← x % 10;
  altfel
    z ← y % 10;
  t ← t + z * u;
  u ← u * 10;
  x ← [x/10];
  y ← [y/10];
cât timp (x ≠ 0 ȘI y ≠ 0);
scrie t;
```

c) Înlocuiți structura de selecție "dacă-altfel" cu una echivalentă, în care atribuirile de pe ramurile "dacă" și "altfel" să fie inversate între ele.

d) Scrieți în pseudocod un algoritm echivalent cu cel dat, în care să se înlocuiască structura "execută...cât timp" cu o structură repetitivă cu test inițial.

Algoritm ► Cel mai mare divizor comun a două numere - algoritmul lui Euclid cu resturi -

Dacă în subcapitolul "Structura repetitivă cu test inițial" v-am prezentat algoritmul lui Euclid cu diferențe, iată că a venit momentul să îl învățați și pe cel cu resturi. Acesta se bazează pe o succesiune de împărțiri, cu extragerea restului la fiecare pas, împărțiri care se reiau până când restul devine zero.

Inițial deîmpărțitul și împărțitorul sunt chiar numerele a și b pentru care trebuie calculat c.m.m.d.c. Proiectăm un ciclu în care la fiecare pas:

- extragem restul împărțirii lui a la b și îl memorăm în variabila r ($r \leftarrow a \% b$);

- actualizăm deîmpărțitul a : noul a va fi împărțitorul anterior ($a \leftarrow b$);

- actualizăm împărțitorul b : noul b va fi restul împărțirii anterioare ($b \leftarrow r$);

Ciclul se repetă până când restul r devine 0. În acel moment algoritmul se încheie, iar cel mai mare divizor comun al celor două numere date este valoarea finală a lui a .

Mai trebuie făcută o precizare: pentru că se împarte a la b , dacă a este mai mic decât b , atunci înaintea declanșării algoritmului trebuie să interschimbăm cele două numere între ele (prin metoda paharelor).

```
citește a, b;
dacă (a < b) atunci
  început
    temp ← a;
    a ← b;
    b ← temp;
  sfârșit
execută
  r ← a % b;
  a ← b;
  b ← r;
cât timp (b ≠ 0);
```

Exemplu de funcționare:

Fie $a=12$ și $b=30$.

$a < b$? $12 < 30$? da $\Rightarrow$ facem interschimbarea valorilor lui a și b între ele:

temp ← a, temp = 12;

a ← b, a = 30;

b ← temp, b = 12.

Având acum $a=30$ și $b=12$, urmărim pașii ciclului:

Pasul 1: $r \leftarrow a \% b$, $r = 30 \% 12$, $r = 6$;

a ← b, a = 12

b ← r, b = 6;

$r = 0$? $6 = 0$? nu $\Rightarrow$ reia ciclul

Pasul 2: $r \leftarrow a \% b$, $r = 12 \% 6$, $r = 0$;

a ← b, a = 6

b ← r, b = 0;

$r = 0$? $0 = 0$? da $\Rightarrow$ încheie ciclul

Cel mai mare divizor comun al numerelor 12 și 30 este ultimul a , adică 6.

Aplicație R.II.21. Multiplii comuni

(Bacalaureat iulie 2008, varianta 34)

Se citesc de la tastatură trei numere naturale nenule x , y și p , având cel mult trei cifre fiecare. Scrieți un algoritm care afișează pe ecran primii p multiplii comuni ai celor două numere x și y .

Exemplu: pentru $x=180$, $y=120$ și $p=5$, se vor afișa numerele: 360, 720, 1080, 1440, 1800.

Rezolvare

Dacă notăm cu m cel mai mic multiplu comun al numerelor x și y , atunci evident primii p multiplii vor fi $m, 2*m, 3*m, \dots, p*m$. Pentru aceasta va trebui mai întâi să calculăm cel mai mare divizor comun al celor două numere x și y , după care vom folosi relația $c.m.m.d.c.(x, y) = (x*y) / c.m.m.d.c.(x, y)$.

În vederea determinării $c.m.m.d.c.(x, y)$, putem folosi algoritmul lui Euclid prezentat mai sus, dar cu o modificare. În ce constă aceasta? Nu putem aplica algoritmul anterior direct pe x și y , pentru că astfel s-ar modifica cele două valori, iar noi avem nevoie ulterior de ele intacte, ca să aflăm $c.m.m.d.c.$ Din această cauză va trebui să folosim niște copii ale numerelor date x și y , pe care le vom nota cu a respectiv b . În loc să inter schimbăm cele două valori între ele în cazul în care prima este mai mică, vom memora de la bun început în a pe cel mai mare dintre numerele x și y , iar în b pe cel mai mic dintre ele. Astfel:

- dacă $x > y$ atunci în a memorăm pe x iar în b pe y ($a \leftarrow x, b \leftarrow y$);
- în caz contrar, a și b vor primi valorile inversate ($a \leftarrow y, b \leftarrow x$).

În continuare aplicăm așa cum am spus algoritmul lui Euclid pentru a și b , iar la finele acestuia memorăm în d cel mai mare divizor comun astfel calculat (care este ultima valoare a variabilei a , adică $d \leftarrow a$). Acum cel mai mic multiplu comun al celor două numere date se obține prin atribuirea $m \leftarrow (x*y) / d$ (produsul celor două numere împărțit la $c.m.m.d.c.$, dar, atenție, nu $m \leftarrow (a*b) / d$, întrucât variabilele a și b nu mai conțin numerele inițiale!).

Odată determinată valoarea lui m reprezentând cel mai mic multiplu comun al lui x și y , nu ne rămâne decât să parcurgem și să afișăm primii p multiplii, care, așa cum am evidențiat la începutul rezolvării, vor fi $m, 2*m, 3*m, \dots, p*m$. În acest sens, avem nevoie de un ciclu, în care contorul i să parcurgă pe rând valorile $1, 2, \dots, p$, și în care la fiecare pas să afișăm un multiplu de forma $i*m$. Dacă alegem o structură repetitivă cu test final, atunci inițializăm $i \leftarrow 1$, iar ciclul arată astfel:

Execută

- afișează multiplul curent, de forma $i*m$, urmat de un spațiu;
- trece la următorul multiplu, prin incrementarea $i \leftarrow i+1$;

Cât timp valoarea lui i este mai mică sau egală cu p .

```

citește x,y,p;
dacă (x>y) atunci
    început
    a←x; b←y;
    sfârșit
altfel

```

```

început
    a←y; b←x;
    sfârșit
execută
    r←a%b;
    a←b;
    b←r;
    cât timp (r#0);
    d←a;
    m←(x*y)/d;
    i←1;
    execută
        scrie i*m, ' ';
        i←i+1;
    cât timp (i<=p);

```

Încercați singuri! - Tema nr. T.21.

► 1. Se citesc de la tastatură trei numere naturale x , y și q . Scrieți un algoritm care determină și afișează pe ecran multiplul comun al lui x și y cel mai apropiat ca valoare de q .

Exemplu: pentru $x=30$, $y=12$ și $q=132$ multiplul căutat este 120, iar pentru $x=30$, $y=12$ și $q=175$, se va afișa multiplul 180.

► 2. (Bacalaureat iulie 2008, varianta 61)

Scrieți un algoritm care citește de la tastatură trei numere naturale x , y și k ($1 \leq x < y \leq 100000$, $k \leq 1000$), apoi afișează pe ecran cele mai mari k numere prime din intervalul $[x, y]$. Valorile respective vor fi tipărite pe un singur rând, separate prin câte un spațiu. Dacă intervalul $[x, y]$ conține mai puțin de k numere prime, atunci se vor afișa pe un rând toate valorile prime găsite separte prin câte un spațiu, iar pe al doilea rând se va scrie mesajul "Insuficiente numere prime".

Exemplu: pentru $x=3$, $y=12$ și $k=5$, ecranul va arăta ca în figură.

```

3 5 7 11
Insuficiente numere prime

```

Ați înțeles? Întrebări cu răspuns scurt

Următoarele două probleme se referă la algoritmul de mai jos, descris în pseudocod, în care s-a notat cu $[a]$ partea întreagă a numărului real a , iar cu $a \% b$ restul împărțirii lui a la b .

```

citește n;
execută
    n←[(n%100)/10]+[n/10];
    cât timp n>=10;
    scrie n;

```

► 1. (Bacalaureat iulie 2009, varianta 59)

Scrieți ce valoare se va afișa, dacă de la tastatură se citește numărul $n=76261$.

► 2. (Bacalaureat iulie 2009, varianta 59)

Găsiți o valoare care ar putea fi citită în variabila n astfel încât să se afișeze rezultatul 6.

Următoarele trei probleme se referă la algoritmul de mai jos, descris în pseudocod, în care s-a notat cu $[a]$ partea întreagă a numărului real a , iar cu $a \bmod b$ restul împărțirii lui a la b .

```
citește x,y;
p←0;
execută
  dacă (y%2≠0) atunci
    p←p+x;
  y←[y/2];
  x←2*x;
  cât timp (y>=1);
  scrie p;
```

► 3. (Bacalaureat iulie 2009, varianta 33)

Scrieți care este valoarea afișată, în cazul în care de la tastatură se citesc numerele: $x=5$ și $y=27$.

► 4. (Bacalaureat iulie 2009, varianta 33)

Găsiți un set de valori nenule care ar putea fi citite în variabilele x și y , astfel încât rezultatul afișat după execuția algoritmului să fie un număr cu cel mult două cifre.

► 5. (Bacalaureat iulie 2009, varianta 33)

Identificați o instrucțiune prin care variabilei p i se atribuie valoarea unei expresii aritmetice echivalentă cu secvență dată (adică rezultatul evaluării expresiei este același cu cel furnizat de către algoritm).

- a) $p \leftarrow a+b$; b) $p \leftarrow a^b$; c) $p \leftarrow a*b$; d) $p \leftarrow a \bmod b$;



II.4.3. Structura repetitivă cu număr fix de pași

Folosim același exemplu pe care am ilustrat celelalte două tipuri de cicluri.

Exemplu:

Să se calculeze suma a n numere naturale citite pe rând de la tastatură (unde n este cunoscut).

Reamintim variantele de rezolvare cu ciclu cu test inițial și final, prezentate în lecțiile anterioare.

```
citește n;
S←0;
i←1;
cât timp (i<=n) execută
  inceput
    citește x;
    S←S+x;
    i←i+1;
  sfârșit
  scrie S;
```

```
citește n;
S←0;
i←1;
execută
  inceput
    citește x;
    S←S+x;
    i←i+1;
  sfârșit
  cât timp (i<=n);
  scrie S;
```

În ambele variante, variabila i va lua pe rând valorile $1, 2, \dots, n$, numărând citirile. Ciclul se oprește după ce s-au făcut n citiri, adică după ce variabila i a ajuns la valoarea n .

La început citim n , apoi inițializăm S cu 0 și i cu 1. În ciclu, la fiecare pas:

- citim un număr în variabila x ;
- adăugăm x -ul citit la suma S ;
- incrementăm cu 1 variabila i ; ea va indica la al câtelea număr am ajuns cu citirea.

Deosebirea dintre cele două variante constă în faptul că în prima testarea condiției are loc la începutul ciclului (care astfel poate să nu se execute niciodată), iar în a doua la sfârșit (după execuția cel puțin a unui pas).

Dar nu putem să precizăm exact câți pași va avea ciclul, fără a mai testa nici o condiție? Răspunsul este afirmativ. Trebuie să comunicăm calculatorului ceva de genul "*execută ciclul pentru $i=1, 2, \dots, n$* ". Această "comandă" poate fi codificată prin intermediul unui **ciclu cu număr fix de pași**, numit și **ciclu cu contor**. Principala parte componentă a unui astfel de ciclu este linia "pentru", prin care specificăm valorile pe care le va primi variabila i . În exemplul nostru, această linie de pseudocod arată astfel:

pentru ($i \leftarrow 1, 2, \dots, n$) execută

Mai notăm că variabila i are denumirea consacrată de *contor al ciclului* deoarece contorizează, adică numără, pașii acestuia.

În această versiune, pseudocodul complet al algoritmului din exemplu se scrie astfel:

```
citește n;
S←0;
pentru ( $i \leftarrow 1, 2, \dots, n$ )
  inceput
    citește x;
    S←S+x;
  sfârșit
  scrie S;
```

În cazul celorlalte două tipuri de structuri repetitive (cu test inițial și cu test final), am impus pseudocodului o sintaxă foarte apropiată de cea a limbajului C++. Aici vom face o excepție. Este vorba despre linia "pentru", pe care în

pseudocod am reprezentat-o într-un mod mai simplist, departe de felul în care trebuie scrisă în programe. În limbajul C++, linia "pentru" are o sintaxă mai complexă, fiind alcătuită din trei expresii: prima expresie trebuie să stabilească valoarea inițială a contorului, a doua va da condiția de continuare a ciclului, iar a treia expresie are rolul de a modifica contorul, dându-i valoarea pe care o va avea la pasul următor.

Deocamdată, chiar și această formă simplificată de pseudocod este suficientă pentru a înțelege "în mare" funcționarea ciclului: contorul i parcurge valorile $1, 2, \dots, n$, și la fiecare pas, pentru fiecare valoare a lui i , se execută corpul ciclului.

Observații:

Să facem o comparație între această variantă și celelalte două variante anterioare.

Ce observăm?

În primul rând remarcăm faptul că cele două instrucțiuni care trebuie să se execute la fiecare pas al ciclului, și anume {citește x ;} și { $S \leftarrow S + x$; } sunt prezente în toate cele trei variante.

În varianta cu ciclu cu număr fix de pași lipsește inițializarea lui i cu 1 dinaintea ciclului ($i \leftarrow 1$), precum și incrementarea lui i ($i \leftarrow i + 1$) din corpul ciclului. De ce? Întrucât într-un ciclu cu contor acestea se fac automat. În momentul în care întâlnește într-un program linia "pentru $i \leftarrow (1, 2, \dots, n)$ execută" (tradusă firește în limbajul de programare folosit), calculatorul "știe" că trebuie să dea lui i valoarea inițială 1 și că la sfârșitul fiecărui pas trebuie să incrementeze pe i cu o unitate.

Pe caz general, nu este obligatoriu ca cele două valori extreme ale contorului să fie 1 și n . În capitolul IV vom vedea că în limbajul C++ ciclurile cu contor admit forme mult mai generale, în care valorile contorului pot fi parcurse în ordine inversă și nu este obligatoriu să fie numere naturale consecutive!

 **Sintaxă:** generală a unui ciclu cu contor este:

pentru ($\langle \text{contor} \rangle \leftarrow \langle v_1 \rangle, \langle v_2 \rangle, \dots, \langle v_n \rangle$)
 $\langle \text{scv} \rangle$

• unde $\langle v_1 \rangle, \langle v_2 \rangle, \dots, \langle v_n \rangle$ sunt valorile succesive ale contorului, iar $\langle \text{scv} \rangle$ este o secvență de instrucțiuni care reprezintă corpul ciclului. La fel ca și în cazul celorlalte două tipuri de ciclu, corpul ciclului cu contor va fi cuprins între cuvintele "început" și "sfârșit" dacă conține mai mult de o instrucțiune.

În exemplul dat, observați că în corpul ciclului contorul nu e folosit în nici un fel, el având strict rolul de a număra pașii ciclului. În continuare veți regăsi numeroase aplicații în care execuția corpului ciclului la fiecare pas este influențată din plin de valoarea contorului.

Reluarea unor algoritmi care au fost prezentați și cu ciclu cu test inițial

Aplicație R.II.22. Număr prim

Scrieți un algoritm care verifică dacă un număr natural x diferit de 0 și 1, este prim sau nu, afișând un mesaj corespunzător.

Precizare: Această problemă a fost rezolvată și în varianta cu ciclu cu test inițial, în cadrul aplicației R.II.18. din lecția "II.4.1. Structura repetitivă cu test inițial".

Rezolvare

 Noțiunea de număr prim împreună cu proprietățile aferente au fost prezentate detaliat în cadrul rezolvării aceleiași probleme în lecția "Structura repetitivă cu test inițial". Un număr natural x este prim, dacă nu are nici un divizor propriu (adică nici un divizor diferit de 1 și el însuși (valorile 1 și x nu intră în discuție, deoarece orice număr natural x este divizibil cu 1 și cu el însuși). Posibili divizori proprii ai lui x se caută printre numerele naturale până la jumătatea lui x . Mai exact, numerele printre care vom căuta divizorii lui x sunt $2, 3, \dots, \lfloor x/2 \rfloor$.

 Deoarece trebuie stabilit dacă este adevărată (îndeplinită) o anumită condiție, avem de-a face cu o testare logică. Este necesară o variabilă cu sens logic, adică o variabilă care poate avea numai două valori, 1 și 0, aferente celor două valori de adevăr posibile, ADEVĂRAT respectiv FALS. Valoarea finală a acestei variabile va indica "starea" condiției testate: 1 dacă numărul x este prim (condiția are valoarea de adevăr ADEVĂRAT), respectiv 0 în caz contrar (condiția are valoarea de adevăr FALS). Denumim această variabilă ok .

 Procedăm prin metoda reducerii la absurd.

- presupunem inițial că numărul x este prim, dând variabilei ok valoarea 1;
- apoi, într-un ciclu cu contor, prin variabila i vor trece pe rând toți posibii divizori ai numărului dat x . Aceștia sunt $2, 3, \dots, x/2$;

✓ la fiecare pas, pentru fiecare valoare a lui i , testăm dacă respectivul i este divizor al lui x , adică dacă restul împărțirii lui x la i este 0 ($x \% i == 0$). În caz afirmativ, numărul x nu mai poate fi prim (am găsit un divizor cert al său, "în persoana" respectivului i), și vom marca acest fapt dând variabilei ok valoarea 0.

 În final se testează valoarea variabilei ok , și în funcție de aceasta se afișează un mesaj din care să reiese dacă numărul x este prim sau nu.

```

citește x;
ok ← 1;
pentru (i ← 2, ..., x / 2)
    dacă (x % i == 0)
        ok ← 0;
dacă (ok == 1)
    scrie 'Numărul este prim'
altfel
    scrie 'Numărul NU este prim';
    
```

Exemplu: Fie $x=15$.

Divizorii acestuia sunt 3 și 5. Pentru descoperirea lor, prin variabila i trec toți posibili divizori, de la 2 la $\lfloor x/2 \rfloor$, deci $i=2, 3, 4, 5, 6, 7$.

Inițial $ok \leftarrow 1$

Pasul 1: cu $i=2$: $x \% i == 0$? $15 \% 2 == 0$? *nu* $\Rightarrow$ nu execută nimic

Pasul 2: cu $i=3$: $x \% i == 0$? $15 \% 3 == 0$? *da* $\Rightarrow ok \leftarrow 0$

După al doilea pas, valoarea 0 a variabilei ok marchează faptul că numărul x nu mai este prim (am găsit pe $i=3$ ca divizor). Pașii următori vor evolua la fel. Pentru $i=5$, variabila ok va mai primi o dată valoarea 0, urmare a faptului că 5 este un nou divizor al lui 15, deci în final valoarea variabilei ok va fi 0.

Dacă în schimb x ar fi avut valoarea 13, atunci condiția " $x \% i == 0$ " nu ar fi fost îndeplinită niciodată, deci variabila ok ar fi rămas cu valoarea inițială 1.

În concluzie, valoarea finală a lui ok ne indică dacă numărul x este prim sau nu. Nu ne rămâne decât să testăm ok și, în funcție de valoarea sa, să afișăm un mesaj sugestiv.

Observație:

Variantă corectă de mai sus nu este optimă. Pe exemplul parcurs, pentru $x=15$, se observă următorul lucru: dovada că numărul x nu este prim s-a obținut deja după pasul al doilea, marcându-se acest fapt prin atribuirea $ok \leftarrow 0$. În consecință, restul pașilor ciclului se execută în mod inutil. Ar trebui ca imediat după ce ok a devenit 0 să se întrerupă ciclul. Algoritm realizat cu o structură "cât timp", prezentat în cadrul aceleiași aplicații în lecția "Structura repetitivă cu test inițial", înlătură această deficiență. Va propunem să vă reamintiți algoritmul optim și să realizați o comparație completă între cele două variante.

Alți algoritmi care folosesc structuri repetitive cu contor

Aplicație R.11.23. Interpretarea unui pseudocod dat

(Bacalaureat iulie 2008, varianta 73)

Se consideră algoritmul alăturat reprezentat în pseudocod, în care toate variabilele identifică numere naturale ($a < b$). S-a notat cu $x \div y$ restul împărțirii numărului natural x la numărul natural nenul y , iar prin $\lfloor x/y \rfloor$ câtul împărțirii întregi a numărului natural x la numărul natural nenul y .

```

citește a, b, p;
nr ← 0;
pentru (i ← a, a+1, ..., b) execută
    început
        x ← i;
        cât timp (x ≠ 0 ȘI x % p ≠ 0) execută
            x ← ⌊x/p⌋;
        dacă (x ≠ 0) atunci
            nr ← nr + 1;
        sfârșit
    scrie nr;

```

a) Scrieți ce se va afișa pe ecran, dacă se citesc pentru a valoarea 24, pentru b valoarea 35 și pentru p valoarea 5.

b) Dacă $a=110$, iar $p=18$, scrieți cel mai mare număr care poate fi citit pentru b , astfel încât să se afișeze valoarea 0.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

d) Scrieți în pseudocod un algoritm echivalent cu cel dat, în care să se înlocuiască structura "cât timp...execută" cu o structură repetitivă de alt tip.

e) Scrieți în pseudocod un algoritm echivalent cu cel dat, care să conțină o singură structură repetitivă, a doua structură repetitivă urmând a fi înlocuită cu o structură liniară.

Rezolvare

a)

Să urmărim evoluția algoritmului pentru $a=24$, $b=35$ și $p=5$.

Inițial $nr=0$. În ciclu, contorul i parcurge succesiv valorile 24, 25, ..., 35, deci ciclul va avea nu mai puțin de 12 de pași. Evident nu o să-i putem detalia pe toți, așa că va trebui să ne dăm seama de la primii pași cam ce se întâmplă.

Pasul 1: $i=24$, $x=i=24$, execută ciclul "cât timp":

Pasul 1.1: $x \neq 0$ ȘI $x \% p \neq 0$? $24 \neq 0$ ȘI $24 \% 5 \neq 0$? *da* $\Rightarrow$

$x \leftarrow \lfloor x/p \rfloor$, $x = \lfloor 24/5 \rfloor$, $x=4$

Pasul 1.2: $x \neq 0$ ȘI $x \% p \neq 0$? $4 \neq 0$ ȘI $4 \% 5 \neq 0$? *da* $\Rightarrow$

$x \leftarrow \lfloor x/p \rfloor$, $x = \lfloor 4/5 \rfloor$, $x=0$

Pasul 1.3: $x \neq 0$ ȘI $x \% p \neq 0$? $0 \neq 0$ ȘI $4 \% 5 \neq 0$? *nu* $\Rightarrow$ iese din ciclu

După ciclul "cât timp", ieșind cu valoarea $x=0$, condiția " $x \neq 0$ " din linia "dacă" nu este îndeplinită.

Pasul 2: $i=25$, $x=i=25$, execută din nou ciclul "cât timp":

Pasul 2.1: $x \neq 0$ ȘI $x \% p \neq 0$? $25 \neq 0$ ȘI $25 \% 5 \neq 0$? *nu* $\Rightarrow$ iese din ciclu

De această dată condiția " $x=0$ " este îndeplinită, motiv pentru care face atribuirea $nr \leftarrow nr+1$, adică $nr=0+1$, $nr=1$.

Pentru a ne da seama ce reprezintă valoarea lui nr , plecăm de la întrebarea "când se incrementează nr ?". Atunci când se iese din ciclul "cât timp" cu o valoare a lui x diferită de zero. În regulă, dar când se întâmplă asta ? Dacă privim derularea algoritmului pe exemplul nostru, prezentată mai sus, observăm că:

– pentru $x=24$, deoarece condiția " $x \% p \neq 0$ " a fost adevărată la primul pas, ea a rămas adevărată la infinit, iar atribuirea $x \leftarrow \lfloor x/p \rfloor$, prin care x se împarte în mod repetat la p , se execută succesiv până când x ajunge la valoarea 0. Practic, ieșirea din ciclu este determinată tocmai de acest moment, în care condiția " $x \neq 0$ " devine falsă. Dacă nu ar exista și condiția " $x \neq 0$ ", am avea de-a face cu un ciclu infinit, întrucât cealaltă condiție, " $x \% p \neq 0$ " este adevărată "în vecii vecilor". De fapt este ceva normal: dacă un număr x nu este divizibil cu p , atunci nici câtul împărțirii lui x la p nu are cum să fie divizibil cu p !

– pentru $x=25$, condiția " $x \% p \neq 0$ " este falsă, chiar de la intrarea în ciclul "cât timp", deci acest ciclu nu va avea nici un pas, iar la finele lui valoarea lui x este cea inițială (adică 25 în cazul nostru). În consecință, condiția " $x \neq 0$ " din linia "dacă" este adevărată, fapt care conduce la incrementarea lui nr . Nu este deloc greu să vă dați seama

că acest lucru se întâmplă pentru toate valorile lui x care sunt divizibile cu p ! În exemplul de la primul subpunct, cu $x=24, 25, \dots, 35$ și $p=5$, incrementarea lui nr se petrece de trei ori, pentru $x=25, x=30$ și $x=35$. În concluzie, algoritmul afișează valoarea 3.

c)

După explicațiile anterioare, credem că v-ați dat seama care este acțiunea algoritmului. El contorizează câte dintre numerele naturale situate în intervalul $[a, b]$ sunt divizibile cu p .

e)

Pornind de la enunțul de mai sus, putem concepe cu ușurință o variantă de algoritm care să nu necesite al doilea ciclu, cel de tipul "cât timp". Practic, se parcurg în contorul i toate numerele naturale de la a la b , și, pentru fiecare valoare a lui i se testează dacă este divizibilă cu p , iar în caz afirmativ se incrementează nr . Iată pseudocodul:

```

citește a, b, p;
nr ← 0;
pentru (i ← a, a+1, ..., b) execută
    început
        x ← i;
        dacă (x%p == 0) atunci
            nr ← nr+1;
        sfârșit
    scrie nr;

```

d)

Practic, putem înlocui structura "cât timp <condiție> adevărată execută <secvență>", cu o structură echivalentă cu test final de genul "execută <secvență> până când <condiție> adevărată".

```

citește a, b, p;
nr ← 0;
pentru (i ← a; a+1, ..., b) execută
    început
        x ← i;
        execută
            x ← [x/p];
        cât timp (x ≠ 0 SI x%p ≠ 0);
        dacă x ≠ 0 atunci
            nr ← nr+1;
        sfârșit
    scrie nr;

```

b)

În cazul $a=110$ și $p=18$, contorul i pleacă de la 110. Pentru ca variabila nr să rămână cu valoarea inițială 0, trebuie ca i să nu ajungă la o valoare divizibilă cu 18 (în momentul în care se întâlnește un număr divizibil cu 18 se incrementează nr !). Cum primul număr natural divizibil cu 18 mai mare ca 110 este 126, rezultă că contorul i poate merge cel mult până la 125. Deci valoarea lui b , care reprezintă celălalt capăt al intervalului, poate fi oricare dintre numerele 110, 111, ..., 125.

Încercați singuri ! - Tema nr. T.24.

► 1. (Bacalaureat iulie 2009, varianta 22)

Se consideră algoritmul alăturat, descris în pseudocod. S-a notat cu $[a]$ partea întreagă a numărului real a și cu $y|x$ faptul că numărul întreg y este divizibil cu numărul întreg nenul x .

a) Scrieți valoarea care se va afișa pentru $n=45$.

b) Scrieți toate valorile cu exact două cifre care pot fi citite pentru variabila n , astfel încât să se afișeze numărul 5.

c) Scrieți o valoare pentru n , astfel încât în urma executării algoritmului obținut prin înlocuirea structurii "pentru ($i \leftarrow 2, 3, \dots, [n/2]$) execută" cu o alta în care contorul parcurge aceleași valori dar în ordine inversă, să se afișeze aceeași valoare, ca în algoritmul inițial.

d) Formulați un enunț scurt care să descrie acțiunea algoritmului.

e) Scrieți un algoritm echivalent, în care structura repetitivă de tipul "pentru" să fie înlocuită cu un alt gen de structură repetitivă.

► 2. (Bacalaureat iulie 2009, varianta 19)

Se consideră algoritmul alăturat, descris în pseudocod, în care s-a notat cu $x \cdot y$ restul împărțirii numărului natural x la numărul natural nenul y și cu $[z]$ partea întreagă a numărului real z .

a) Scrieți numerele care se vor afișa în urma executării algoritmului, dacă se citesc valorile $a=312$ și $b=1354$.

b) Scrieți câte o valoare care poate fi citită pentru variabila a , respectiv b , astfel încât algoritmul să afișeze exact două valori.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

d) Scrieți în pseudocod un algoritm echivalent cu cel dat, în care să se înlocuiască structura "pentru... execută" cu o structură repetitivă de alt tip.

► 3. (Bacalaureat iulie 2009, varianta 71)

Se consideră algoritmul alăturat, descris în pseudocod. S-a notat cu $[a/b]$ câtul împărțirii întregi a numărului natural a la numărul natural nenul b .

a) Scrieți valoarea ce se va afișa dacă se citesc pentru n valoarea 5 și pentru x , în această ordine, valorile: 523, 4256, 324, 4, 86935.

```

citește n;
dacă (n < 0) atunci
    n ← -n;
d ← 1;
pentru (i ← 2, 3, ..., [n/2]) execută
    dacă (i | n) atunci
        d ← i;
    scrie d;

```

```

citește a, b;
a ← [a/10]*10 + a%10;
b ← [b/10]*10 + b%10;
pentru (i ← a, a+1, ..., b) execută
    dacă ([i/10] = i%10) atunci
        scrie i%10;

```

b) Dacă $n=4$, scrieți un șir de valori care pot fi citite pentru x astfel încât să se afișeze 1234.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului.

d) Scrieți în pseudocod un algoritm echivalent cu cel dat, care să utilizeze în locul structurii "cât timp...execută" o structură repetitivă condiționată posterior.

```
s ← 0;
citește n;
pentru (i ← 1, 2, ..., n) execută
    început
        citește x;
        cât timp (x > 9) execută
            x ← [x/10];
        pentru (j ← 1, 2, ..., i-1) execută
            x ← x*10;
        s ← s+x;
    sfârșit
scrie s
```

► 4. (Bacalaureat iulie 2000, varianta 4)

Se consideră următoarea secvență de instrucțiuni, scrisă în pseudocod:

a) Ce se va afișa pe ecran dacă se citesc în ordine valorile: 9 1 1 1 2 2 3 3 3 3?

b) Scrieți o secvență de instrucțiuni echivalentă care să utilizeze o structură repetitivă cât timp.

c) Formulați un enunț scurt care să descrie acțiunea algoritmului dat.

```
citește n, a;
nr ← 1; max ← 1;
pentru (i ← 2, 3, ..., n) execută
    început
        citește b
        dacă (a == b) atunci
            nr ← nr + 1
        altfel
            început
                dacă (nr > max) atunci
                    max ← nr
            nr ← 1
        sfârșit
    a ← b
    sfârșit
scrie max
```

Aplicație R. II. 24. Numere pare și divizibile cu 5

(Bacalaureat iulie 2008, varianta 27, enunț adaptat)

Scrieți un program care, pentru un număr natural n nenul de cel mult patru cifre citit de la tastatură, afișează, în ordine crescătoare primele n numere naturale pare și divizibile cu 5, precum, și suma acestora.

Exemplu: pentru $n=6$ se vor afișa numerele 10 20 30 40 50 60

Rezolvare

Pentru început trebuie să citim valoarea lui n , și să inițializăm cu 0 variabila s în care vom actualiza suma numerelor cerute.

După cum știți, numerele pare și divizibile cu 5, fiind divizibile cu 2 și 5, sunt în mod categoric divizibile și cu 10. Așadar problema se reduce la generarea primelor n numere naturale divizibile cu 10. Acestea sunt de forma $10 \cdot k$, unde k este un număr natural ($10=10 \cdot 1$, $20=10 \cdot 2$, $30=10 \cdot 3$, etc.). Întrucât se cer primele n astfel de numere,

avem nevoie de un ciclu "pentru", în care contorul k va lua pe rând valorile $1, 2, \dots, n$. La fiecare pas:

- afișăm un număr de forma $10 \cdot k$ (cel rezultat pentru valoarea lui k de la acel pas), urmat de un spațiu;
- adăugăm valoarea în cauză la suma s a tuturor numerelor generate $\{s \leftarrow s + 10 \cdot k\}$.

În final afișăm suma s a primelor n numere pare și divizibile cu 5.

```
citește n;
s ← 0;
pentru (k ← 1, 2, ..., n) execută
    început
        scrie 10*k;
        s ← s + 10*k;
    sfârșit
scrie s;
```

Încercați singuri! - Tema nr. T.24.

► 1. Fiind date două numere întregi a și m , să se scrie o funcție care returnează valoarea lui a^m , calculată prin înmulțiri repetate, după formula $a^m = a \cdot a \cdot \dots \cdot a$ (de m ori).

► 2. Să se calculeze produsul a două numere prin adunări repetate, după relația $a \cdot b = a + a + \dots + a$ (de b ori).

► 3. Fiind dat un număr natural n , citit de la tastatură, să se afișeze produsul numerelor naturale mai mici sau egale decât n (factorialul numărului natural n , notat $n! = 1 \cdot 2 \cdot \dots \cdot n$).

Aplicație R. II. 25. Șirul lui Fibonacci

Să se afișeze primii n termeni ai șirului lui Fibonacci. Șirul are primii doi termeni egali cu 1 și fiecare din următorii termeni este egal cu suma dintre termenul precedent și termenul ante-precedent.

Rezolvare

Fie $F_1, F_2, \dots, F_n$ primii n termeni ai șirului. Conform definiției avem: $F_1 = 1, F_2 = 1, F_k = F_{k-1} + F_{k-2} \quad \forall k \geq 3$

De exemplu: pentru $k=3 \Rightarrow F_3 = F_2 + F_1 = 1 + 1 = 2$
pentru $k=4 \Rightarrow F_4 = F_3 + F_2 = 2 + 1 = 3$

Generăm termenii șirului începând cu al treilea ($F_3, F_4, \dots, F_n$) într-un ciclu: indicele i ia pe rând valorile $3, 4, \dots, n$, și pentru fiecare i se generează termenul F_i în variabila `fib`. Memorăm precedentul și ante-precedentul fiecărui termen în variabilele `prec` și `anteprec`.

fib prec anteprec
 $F_3 = F_2 + F_1$
 F₄ = F₃ + F₂
 noul noul
 prec anteprec

- noul ante-precedent va fi cel care a fost precedent la pasul curent adică **prec** (**anteprec←prec;**)
- noul precedent pentru pasul următor va fi termenul tocmai generat, adică **fib** (**prec←fib**)

```

citește n;
prec ← 1;
anteprec ← 1;
scrie prec, anteprec;
pentru (i ← 3, ..., n) execută
    început
        fib ← prec + anteprec;
        scrie fib;
        anteprec ← prec;
        prec ← fib;
    sfârșit

```

Încercați singuri ! - Tema nr. T.25.

$$a_{n+1} = 1 + a_n^2 \quad (\forall) n \geq 1, \quad a_0 = 1$$
$$x_n = 2 \cdot x_{n-1} + x_{n-2} \quad (\forall) n \geq 2, x_0 = 0, x_1 = 1$$

Aplicație R.II.26. Triunghi de numere

$$\begin{array}{ccccccc} 1 & & & & & & \\ 1 & 3 & & & & & \\ 1 & 3 & 5 & & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 3 & 5 & \dots & \dots & \dots & 2n-1 \end{array}$$

Rezolvare

82

Să vedem cum afișăm o linie i . Observăm că pe linia 1 avem un element, pe linia a doua două elemente, pe linia a treia trei elemente etc., deci pe o linie oarecare i avem i elemente. Numerotăm elementele de pe linia i cu $1, 2, \dots, i$. Folosim un alt ciclu "pentru", în care valorile contorului j sunt numerele de ordine ale elementelor de pe linia i , adică $j=1, 2, \dots, i$; la fiecare pas, afișăm elementul cu numărul de ordine j . Acesta este de forma $2*j-1$, după cum se poate observa din exemplul de mai jos în care considerăm linia $i=3$ cu elementele 1, 3, 5.

- ✓ $j=1 \Rightarrow$ elementul cu numărul de ordine 1 este $1=2*1-1$
- ✓ $j=2 \Rightarrow$ elementul cu numărul de ordine 2 este $3=2*2-1$
- ✓ $j=3 \Rightarrow$ elementul cu numărul de ordine 3 este $5=2*3-1$

```

citește n;
pentru (i←1,...,n) execută
    început
        pentru (j←1,...,i) execută
            scrie. 2*j-1, ' ';
        <salt la rând nou>;
    sfârșit;
sfârșit

```

Încercați singuri ! - Tema nr. T.26.

$$\begin{array}{ccccccc} 2 & & & & & & \\ 2 & 4 & & & & & \\ 2 & 4 & 6 & & & & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 2 & 4 & 6 & \dots & \dots & \dots & 2n \end{array}$$

2
2 4
2 4 6
2 4 6 8

```

1 2 3 ..... n
.....
1 2 3
1 2
1

```

► 1. (Bacalaureat iulie 2008, varianta 27)

Algoritmul de mai jos își propune să determine în variabila min cea mai mică valoare dintre cele n numere întregi citite de la tastatură, numere care sunt mai mici decât 100. Precizați valoarea inițială a variabilei min , care trebuie scrisă în loc de "...", astfel încât algoritmul să aibă efectul dorit.

```

citește n;
min ← ...;
pentru i de la 1 la n execută
    început
        citește x;
        dacă x < min atunci min ← x;
    sfârșit;
scrie min;

```

- a) 100 b) 1 c) 0 d) -MAXINT

► 2. (Bacalaureat iulie 2008, varianta 45)

Deduceți care este valoarea afișată în urma execuției algoritmului reprezentat prin pseudocodul de mai jos:

```

y ← 0; x ← 0;
pentru (i ← 1, 2, 3) execută
    început
        x ← y + i;
        y ← x;
    sfârșit;
scrie x;

```

Următoarele două probleme se referă la algoritmul de mai jos reprezentat în pseudocod, în care toate variabilele identifică numere naturale, iar prin $a \bmod b$ s-a notat restul împărțirii lui a la b .

```

citește x;
p ← 1;
pentru (i ← 1, 2, ..., x) execută
    p ← (p * 4) % 10;
scrie p;

```

► 3. (Bacalaureat iulie 2009, varianta 99)

Deduceți ce valoare se va afișa în urma execuției algoritmului, dacă de la tastatură se citește $x=8$.

► 4. (Bacalaureat iulie 2009, varianta 99)

Scrieți toate numerele naturale alcătuite dintr-o singură cifră care ar putea fi citite în variabila x , astfel încât, la finele execuției algoritmului, să se tipărească valoarea 4.

Următoarele două probleme se referă la algoritmul de mai jos reprezentat în pseudocod, în care toate variabilele identifică numere naturale.

```

citește n;
k ← 0;
pentru (i ← 1, 2, ..., n) execută
    pentru (j ← 1, 2, ..., i) execută
        început
            scrie i + j;
        k ← k + 1;
    sfârșit;
scrie k;

```

► 5. (Bacalaureat iulie 2009, varianta 64)

Dacă de la tastatură se citește valoarea $n=4$, care este antepenultimul număr afișat.

► 6. Dacă de la tastatură se citește valoarea $n=7$, câte numere se afișează în total pe parcursul execuției algoritmului?

► 7. (Bacalaureat iulie 2009, varianta 64)

Deduceți ce valoare ar putea fi citită în variabila n , astfel încât numărul 64 să fie afișat de exact 20 de ori.



Probleme propuse

Tema nr. T.R.3.

Probleme propuse pentru toate tipurile de cicluri

► 1. Fiind dat un număr natural n , scrieți un algoritm pentru calculul expresiei $E = 1*2 + 2*3 + 3*4 + \dots + n*(n+1)$.

► 2. (Bacalaureat iulie 2009, varianta 11)

Scrieți un algoritm care citește de la tastatură un număr natural nenul n , precum și n numere naturale, cu maximum nouă cifre fiecare, apoi calculează și afișează pe ecran, câte numere prime conține șirul citit.

Exemplu: pentru $n=5$ și valorile 12 3 9 7 1 se va afișa pe ecran valoarea 2 (în șirul dat există două numere prime și anume 3 și 7).

Indicații: Citim numerele în aceeași variabilă x ; într-un ciclu în care contorul i , cu valorile $1, 2, \dots, n$, nu are rol decât acela de a numără citirile. Pentru fiecare x citit, testăm dacă este prim cu algoritmul cunoscut, iar în caz afirmativ incrementăm un contor care numără valorile prime (inițializat cu 0).

► 3. (Bacalaureat iulie 2008, varianta 77)

Se citesc de la tastatură un număr natural n și un șir de n numere întregi. Scrieți un algoritm eficient care determină cea mai mare valoare negativă dintre cele citite, precum și numărul de apariții ale acesteia în cadrul șirului. În cazul în care șirul citit nu conține nici o valoare negativă, se va afișa mesajul "NU EXISTĂ".

Exemplu: Pentru $n=8$ și numerele $(10, 3, -4, -7, -4, -9, 11, -4)$, se vor afișa valorile -4 și 3 , deoarece -4 este cel mai mare element negativ al șirului și apare de 3 ori.

► 4. (Bacalaureat iulie 2008, varianta 53)

Se citește de la tastatură două numere naturale n și p , ($1 < n < 1000, 1 < p < 10$). Să se afișeze pe ecran, cu spațiu între ele, acele numere naturale mai mici sau egale cu n , care au toate cifrele mai mici sau egale cu p .

Exemplu: pentru $n=15$ și $p=2$, se vor afișa numerele $0, 1, 2, 10, 11, 12$.

► 5. Realizați un algoritm care afișează "cifra de control" a unui număr natural, determinată astfel: se calculează suma cifrelor numărului, apoi suma cifrelor sumei obținute, ș.a.m.d., până când rezultatul este o singură cifră.

► 6. Două numere naturale a și b se numesc "gemene" dacă sunt prime și diferența lor în modul este 2. Să se afișeze perechile de numere "gemene" care se pot forma pe mulțimea numerelor naturale mai mici sau egale cu n , unde valoarea lui n se citește de la tastatură. Se cere un algoritm eficient de rezolvare.

Indicații: Pentru a evita testarea condiției privitoare la diferența în modul este suficient să analizăm perechile de numere de forma $(i, i+2)$ cu $i=1, 2, \dots, n-2$, și pentru fiecare pereche să testăm dacă i și $i+2$ sunt ambele numere prime. Realizarea unui algoritm eficient presupune:

- să nu se parcurgă divizorii inutili (vezi varianta cu ciclul cu test inițial a algoritmului de număr prim);
- să nu se mai testeze $i+2$ dacă i nu este prim.

► 7. Se citește de la tastatură un număr natural n ($100 \leq n \leq 999$). Dintre numerele naturale mai mici sau egale cu n , să se afișeze acelea care sunt divizibile cu suma cifrelor lor.

► 8. Dintre primele n numere naturale, să se afișeze acelea care au exact k divizori. Valorile lui n și k se citesc de la tastatură.

► 9. (Bacalaureat iulie 2009, varianta 97 - enunț adaptat)

Se citesc de la tastatură două numere naturale x și y , cu maxim patru cifre fiecare. Scrieți un algoritm care tipărește pe ecran suma numerelor pare cuprinse între x și y inclusiv.

Exemplu: pentru $x=12$ și $y=23$, se va afișa valoarea 102.

► 10. Să se afișeze toate numerele perfecte situate în intervalul $[p, q]$, precum și numărul acestora, unde p și q sunt două numere naturale date (vezi problema 4).

► 11. Să se afișeze numerele prime de trei cifre care citite invers sunt tot numere prime.

► 12. Se dau două numere naturale n și p . Să se stabilească dacă p este prim și în caz afirmativ să se afișeze la ce putere apare p ca factor prim în descompunerea numărului n în factori primi.

► 13. Un număr natural se numește *perfect* dacă este egal cu suma divizorilor săi. *Exemplu:* $6=1+2+3$. Să se verifice dacă un număr natural dat este perfect, afișându-se un mesaj corespunzător.

► 14. Într-o urnă se găsesc mai multe bile roșii și albe, pe fiecare bilă fiind înscris un număr natural. Să se extragă pe rând bile, până când am scos din urnă cel puțin m bile albe sau n bile roșii. Scrieți un program care simulează această operație, citind de la tastatură numărul fiecărei bile extrase. În final, programul va afișa dacă cel mai mic număr extras se găsește pe o bilă albă sau roșie.

► 15. Se citește de la tastatură un întreg n , apoi n perechi de numere naturale cuprinse între 0 și 30000. Dintre perechile citite să se afișeze acelea cu proprietatea că cele două elemente ale perechii au aceeași sumă a cifrelor.

► 16. (Bacalaureat iulie 2008, varianta 36)

Realizați un algoritm care afișează pe ecran toate numerele pare alcătuite din exact două cifre. Respectiv valorile vor fi scrise pe mai multe linii de ecran, câte cinci pe fiecare linie, separate prin câte un spațiu.

► 17. Se dau două numere naturale n și p . Să se stabilească dacă p este prim și în caz afirmativ să se afișeze la ce putere apare p ca factor prim în descompunerea numărului n în factori primi.

► 18. (Bacalaureat iulie 2008, varianta 41)

Realizați un algoritm care afișează pe ecran toate numerele impare alcătuite din exact două cifre. Respectiv valorile vor fi scrise pe mai multe linii de ecran, câte trei pe fiecare linie, separate prin câte un spațiu.

► 19. (Bacalaureat iulie 2009, varianta 62)

Scrieți un algoritm care afișează pe ecran primii 20 de termeni impari ai șirului lui Fibonacci $(0, 1, 1, 2, 3, 5, 8, 13, 21, \dots)$, câte 5 pe o linie. Termenii afișați pe aceeași linie a ecranului vor fi separați între ei prin câte un spațiu.

► 20. Se dă un șir de numere care se citesc pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din șir). Să se afișeze perechile de numere consecutiv citite cu proprietatea că al doilea element al perechii este egal cu dublul primului.

► 21. Pentru un număr întreg n dat, să se calculeze fiecare dintre expresiile de mai jos:

a) $E = 1^2 + 4^2 + 7^2 + \dots + (3n - 2)^2$

b) $E = 1 \cdot 3 \cdot 5 \cdot \dots \cdot (2n - 1)$

c) $E = 1 + 1 \cdot 2 + 1 \cdot 2 \cdot 3 + \dots + 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$

d) $E = 2 - 4 + 6 + \dots + (-1)^{n+1} \cdot (2n)$

► 22. Se citesc de la tastatură două numere întregi x și n . Calculați valoarea sumei: $s = x + \frac{x^3}{3} + \frac{x^5}{5} + \dots + \frac{x^{2n-1}}{(2n-1)}$

► 23. Un număr natural se numește *perfect* dacă este egal cu suma divizorilor săi. *Exemplu:* $6=1+2+3$. Să se verifice dacă un număr natural dat este perfect, afișându-se un mesaj corespunzător.

► 24. Într-o urnă se găsesc mai multe bile roșii și albe, pe fiecare bilă fiind înscris un număr natural. Să se extragă pe rând bile, până când am scos din urnă cel puțin m bile albe sau n bile roșii. Scrieți un program care simulează această operație, citind de la tastatură numărul fiecărei bile extrase. În final, programul va afișa dacă cel mai mic număr extras se găsește pe o bilă albă sau roșie.

► 25. Se citește de la tastatură un întreg n , apoi n perechi de numere naturale cuprinse între 0 și 30000. Dintre perechile citite să se afișeze acelea cu proprietatea că cele două elemente ale perechii au aceeași sumă a cifrelor.

► 26. (Bacalaureat iulie 2009, varianta 87 - enunț adaptat)

Se citesc de la tastatură două numere naturale a și b unde a conține cel mult nouă cifre, iar b este strict mai mic decât numărul de cifre a lui a . Scrieți un algoritm care înlocuiește cu 0 ultimele b cifre ale numărului a și afișează pe ecran valoarea astfel obținută.

Exemplu: pentru $a=1844657$ și $b=3$, numărul tipărit este 1844000.

Indicații: Trebuie să determinăm câtul împărțirii lui a la 10^b , iar pe acesta să-l înmulțim cu 10^b . De pildă, pentru $a=1844657$ și $b=3$, avem $10^3=1000$, apoi $1844657 \div 1000=1844$ și $1844 \cdot 1000=1844000$.

► 27. (Bacalaureat iulie 2008, varianta 79)

Scrieți un algoritm care elimină prima și ultima cifră dintr-un număr natural n citit de la tastatură. Numărul dat poate avea minim 3 și maxim nouă cifre.

Exemplu: pentru $n=323958$, se va afișa pe ecran numărul 2395.

► 28. (Bacalaureat iulie 2009, varianta 97)

Se citește de la tastatură un număr natural n cu maxim nouă cifre. Scrieți un algoritm care afișează pe ecran numărul care se obține cu cifrele de rang impar ale lui n luate în ordine naturală, urmate de cifrele de rang par luate în ordine inversă. Prin rangul unei cifre înțelegem poziția pe care o ocupă cifra în cadrul numărului, numărată de la dreapta la stânga.

Exemplu: pentru $n=275839$, algoritmul va tipări numărul 789352.

► 29. (Bacalaureat iulie 2008, varianta 98)

Se citește de la tastatură un număr natural n alcătuit din cel mult opt cifre. Scrieți un algoritm care afișează mesajul "DA" dacă numărul dat are toate cifrele identice, respectiv "NU" în caz contrar.

Exemplu: pentru $n=77777$, se va afișa mesajul "DA"

► 30. (Bacalaureat iulie 2008, varianta 44)

Se citesc de la tastatură două numere naturale n și k , unde n poate avea maxim nouă cifre și $0 < k < 10$. Scrieți un algoritm care afișează numărul obținut prin eliminarea primelor k cifre ale lui n . Dacă valoarea lui k este mai mare decât numărul de cifre ale lui n , se va afișa mesajul "IMPOSIBIL".

► 31. (Bacalaureat iulie 2009, varianta 76 - enunț adaptat)

Scrieți un algoritm care citește de la tastatură un număr natural m cu maxim nouă cifre și o cifră zecimală c aparținând mulțimii $\{0, 1, 2, \dots, 9\}$, apoi determină și afișează numărul de cifre ale lui n care aparțin intervalului $[c-1, c+1]$.

Exemplu: pentru $n=1233$ și $c=3$ se va afișa valoarea 3, iar pentru $n=650$ și $c=3$ algoritmul trebuie să tipărească valoarea 0.

► 32. (Bacalaureat iulie 2009, varianta 89 - enunț adaptat)

Se citesc de la tastatură valorile a trei variabile n , $c1$ și $c2$, unde n este un număr natural format din cel mult nouă cifre, iar $c1$ și $c2$ memorează câte o cifră. Realizați un algoritm care înlocuiește cu cifra $c2$ fiecare apariție a cifrei $c1$ în cadrul numărului n , și afișează pe ecran valoarea astfel obținută. În cazul în care cifra $c1$ nu se regăsește în numărul n , valoarea lui n va rămâne nemodificată.

Exemplu: pentru $n=149448$, $c1=4$ și $c2=2$ algoritmul va afișa valoarea 129228.

► 33. (Bacalaureat iulie 2009, varianta 27 - enunț adaptat)

Scrieți un algoritm care citește de la tastatură două numere naturale x și y , cuprinse în intervalul $[1, 100]$, apoi formează în memorie numărul real ce are proprietatea că partea sa întreagă este egală cu x , iar cifrele aflate după punctul zecimal sunt cifrele numărului y în aceeași ordine.

Exemplu: pentru $x=12$ și $y=534$, pe ecran trebuie să se afișeze numărul 12.534.

Indicații: Mai întâi formăm în memorie numărul $z=0.y$, adică numărul care are partea întreagă 0 și partea zecimală alcătuită din cifrele lui y . Pentru aceasta, împărțim în mod repetat pe y la 10, până când câtul împărțirii devine mai mic decât 1. Numărul căutat va fi $x+z$. Pentru exemplul din enunț, cu $x=12$ și $y=534$, avem $z=0.534$, iar $x+z=12+0.534=12.534$.

► 34. (Bacalaureat iulie 2009, varianta 79 - enunț adaptat)

Se citește de la tastatură un număr real x , care poate conține cel mult patru cifre în partea întreagă și cel mult șase cifre după punctul zecimal. Realizați un algoritm care construiește în memorie și afișează pe ecran numărul obținut prin inversarea între ele a părții întregi și a părții zecimale a părții lui x .

Exemplu: dacă se citește valoare $x=15.231$, trebuie să se afișeze numărul 231.15.

► 35. (Bacalaureat iulie 2008, varianta 15)

Se citește de la tastatură un număr natural n ($n < 100$), apoi n numere naturale din mulțimea $\{1, 2, 3\}$. Se cere să se afișeze cele n valori citite în ordine crescătoare, folosind un algoritm eficient din punctul de vedere al timpului de execuție și al spațiului de memorare utilizat.

Exemplu: pentru $n=19$ și valorile citite 3, 3, 3, 3, 2, 1, 2, 1, 3, 2, 1, 3, 2, 1, 1, 3, 3, 2, 3, se va afișa pe ecran șirul 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3.

Indicații: Folosim trei contoare $c1$, $c2$, $c3$, în care vom număra valorile de 1, 2, respectiv 3 din șir. Citim pe rând cele n valori în aceeași variabilă x , într-un ciclu în care contorul $i=1, 2, \dots, n$ numără citirile. Pentru fiecare x citit, în funcție de valoarea sa, incrementăm contorul corespunzător. În final, folosim trei cicluri distincte, afișăm numărul 1 de $c1$ ori, valoarea 2 de $c2$ ori și numărul 3 de $c3$ ori.

►36. Să se determine toate numerele de trei cifre cu proprietatea că suma inverselor cifrelor lor este un număr subunitar, iar cifrele sunt în ordine strict crescătoare. *Exemplu:* Numărul 245: $1/5 + 1/4 + 1/2 = \text{fracție subunitară}$.

Indicații: se generează toate posibilitățile de a scrie cifre în ordinea crescătoare și se verifică dacă suma inverselor cifrelor este un număr subunitar. În caz afirmativ se va afișa numărul format cu cele trei cifre.

►37. Se citește un număr natural n și coeficienții a, b, c , ai unei ecuații de gradul II de forma $ax^2 + bx + c = 0$. Fără a rezolva ecuația, să se calculeze valoarea expresiei $S_n = x_1^n + x_2^n$, unde x_1 și x_2 sunt rădăcinile ecuației.

Indicații: Avem $S_1 = x_1 + x_2 = -b/a$ (relația lui Viète). Deoarece x_1 este rădăcină, rezultă că $ax_1^2 + bx_1 + c = 0$ (1), și analog, $ax_2^2 + bx_2 + c = 0$ (2). Adunând cele două relații obținem $aS_2 + bS_1 + c = 0$. Mai departe înmulțim relația (1) cu x_1 și relația (2) cu x_2 după care adunăm, rezultând $aS_3 + bS_2 + cS_1 = 0$. Generalizând, vom vedea $aS_n + bS_{n-1} + cS_{n-2} = 0$, recurență care se poate calcula într-un ciclu.

►38. Să se calculeze puterea la care apare numărul prim p în descompunerea numărului $1 \cdot 2 \cdot 3 \cdot \dots \cdot n$, pentru un n natural dat.

Exemplu: pentru $n=36$ și $p=3$, se va obține puterea 16.

Indicații. Facem mai întâi observația că pentru un n foarte mare, este imposibilă memorarea produsului $1 \cdot 2 \cdot 3 \cdot \dots \cdot n$ ca număr întreg. De aceea, rezolvarea trebuie să evite folosirea acestui produs. Dacă p este număr prim, atunci din " p divide $a \cdot b$ " rezultă " p divide a " sau " p divide b ". În consecință, vom determina puterea lui p în dezvoltarea fiecărui factor al produsului $1 \cdot 2 \cdot 3 \cdot \dots \cdot n$ și vom însuma puterile obținute. Cum p este prim, vom lua în considerare numai factorii de forma $k \cdot p$, unde k va lua valori de la 1 până la "parte întreagă din n/p ".

►39. (Bacalaureat iulie 2008, varianta 51)

Se citește de la tastatură un număr natural nenul n cu cel mult nouă cifre. Scrieți un algoritm care găsește și afișează un număr natural k astfel încât produsul $1 \cdot 2 \cdot 3 \cdot \dots \cdot k - 1 \cdot k$ să aibă o valoare cât mai apropiată de n .

Exemplu: pentru $n=25$ se va obține $k=4$ ($1 \cdot 2 \cdot 3 \cdot 4 = 24$), iar pentru $n=119$ programul va afișa numărul 5.

►40. (Bacalaureat iulie 2008, varianta 17)

Scrieți un algoritm care afișează pe ecran, separate prin virgulă, toate numerele naturale de patru cifre care au prima cifră strict mai mică decât a doua, a doua cifră strict mai mică decât a treia, și a treia cifră strict mai mică decât a patra.

Exemplu: câteva astfel de numere ar fi 1201, 1202, 1203, ..., 8979, 8989.

Indicații: pentru a obține toate numerele de forma $c_1c_2c_3c_4$ care să respecte monotonia dorită, adică $c_1 < c_2$, $c_2 < c_3$ și $c_3 < c_4$, putem proiecta patru cicluri imbricate, în care contoarele vor lua valorile $c_1 = 1, \dots, 8$, $c_2 = c_1 + 1, \dots, 9$, $c_3 = 0, \dots, c_2 - 1$ și $c_4 = c_3 + 1, \dots, 9$. Numerele căutate sunt de forma $c_1 \cdot 1000 + c_2 \cdot 100 + c_3 \cdot 10 + c_4$.

►41. (Bacalaureat iulie 2008, varianta 61)

Se citește de la tastatură un număr natural k ($1 \leq k \leq 4$). Să se afișeze toate palindroamele de cinci cifre (adică numerele naturale de forma $c_1c_2c_3c_4c_5$ pentru care $c_1 = c_5$ și $c_2 = c_4$) cu proprietatea că diferența în valoare absolută dintre

oricare două cifre alăturate este egală cu k (adică $|c_i - c_{i+1}| = k$, pentru orice $i \in \{1, 2, 3, 4\}$).

Indicații: singurele numere care satisfac proprietatea cerută au una din formele $(i, i-k, i, i-k, i)$ $(i, i-k, i-2 \cdot k, i-k, i)$ $(i, i+k, i, i+k, i)$ $(i, i+k, i+2 \cdot k, i+k, i)$. Folosim un ciclu în care contorul i va primi ca valori cifrele zecimale de la 1 la 9; pentru fiecare i , afișăm numerele ce se încadrează în formele de mai sus, având însă grijă să punem și condițiile de existență a lor: $i-k \geq 0$, $i-2 \cdot k \geq 0$, $i+k \leq 9$, $i+2 \cdot k \leq 9$.

►42. (Bacalaureat iulie 2008, varianta 78)

Scrieți un program Pascal / C/C++ care citește de la tastatură un număr natural n ($1 < n < 30000$) și care determină și afișează pe ecran numărul divizorilor primi a lui n .

Exemplu: pentru $n=24$ se afișează 2 (deoarece are 2 divizori primi: 2 și 3), iar pentru numărul 60 se afișează 3 (deoarece are 3 divizori primi: 2, 3 și 5).

►43. (Bacalaureat iulie 2008, varianta 60)

Se citește de la tastatură trei numere naturale de cel mult opt cifre fiecare n , a și b și se cere să se afișeze pe ecran câte numere naturale mai mici sau egale cu n sunt multipli a lui a , dar nu sunt multipli a lui b . De exemplu, dacă $n=100$, $a=12$, $b=8$, rezultatul afișat va fi 4 (numerele mai mici sau egale cu 100 care sunt multipli a lui 12 dar nu și a lui 8 sunt 12, 36, 60, 84).

a) Alegeți o metodă eficientă de rezolvare și descrieți în limbaj natural metoda aleasă justificând eficiența acesteia (4-6 rânduri)

b) Scrieți programul Pascal / C/C++ corespunzător metodei alese la punctul a.

►44. (Bacalaureat iulie 2008, varianta 62)

a) Scrieți un program în limbajul Pascal / C/C++ , eficient din punctul de vedere al duratei de executare, care afișează toate numerele naturale formate din cifre identice, mai mari decât 10 și mai mici decât o valoare dată n , $n \leq 2.000.000.000$.

De exemplu pentru $n=195$, se afișează: 11, 22, 33, 44, 55, 66, 77, 88, 99, 111.

b) Explicați în limbaj natural metoda utilizată, justificând eficiența ei (4-6 rânduri).

►45. (Bacalaureat iulie 2008, varianta 64)

Scrieți un program Pascal / C/C++ care citește de la tastatură două numere naturale nenule n și m ($n < 100$, $m < 100$), determină și afișează pe ecran toate perechile a, b de numere naturale nenule distincte, cel mult egale cu n , care au cel mai mic multiplu comun egal cu m . Perechile se vor afișa pe câte o linie, iar numerele corespunzătoare fiecărei perechi vor fi separate printr-un spațiu, ca în exemplu.

De exemplu, pentru $n=6$ și $m=6$ se afișează perechile:

(1 6) (2 3) (2 6) (3 6)

►46. (Bacalaureat iulie 2008, varianta 78)

Se citește de la tastatură mai multe numere naturale de cel mult 3 cifre. Se știe că se citește cel puțin 2 numere și că citirea continuă, număr cu număr, până când se citește un număr egal cu primul număr citit. Să se scrie un program Pascal, C/C++ care citește numerele și afișează pe ecran media aritmetică a numerelor

pare citite precum și numărul acestora. Se va afișa mesajul NU EXISTĂ dacă nu există nici un număr par printre cele citite. Alegeți o metodă de rezolvare care va gestiona eficient memoria.

De exemplu, dacă se citesc pe rând numerele 5 6 7 9 6 8 5 se vor afișa pe ecran valorile 5.5 și 4.

► 47. (Bacalaureat iulie 2008, varianta 95)

Scrieți un program care citește de la tastatură două numere naturale a și b ($0 \leq a, b \leq 2000000000$) și în cazul în care există cel puțin o aceeași cifră în ambele numere, afișează mesajul DA, iar în caz contrar afișează mesajul NU.

De exemplu, pentru $a=83$ și $b=119$ se va afișa pe ecran mesajul NU, iar pentru $a=5003$ și $b=720$ se va afișa DA.

► 48. Se citesc de la tastatură mai multe șiruri de numere întregi, fiecare șir terminându-se cu valoarea 0. Citirea se încheie prin introducerea de două ori consecutive a valorii 0. Să se afișeze elementul maxim din fiecare șir, precum și maximul tuturor șirurilor.

Indicații: Pentru a asigura încheierea citirii la întâlnirea a două zerouri consecutive procedăm astfel: în ciclul de citire, înainte de citirea unui număr în variabila x , salvăm x -ul citit la pasul anterior într-o variabilă $prec$ cu sensul de "termen precedent" (inițial citim primul x înainte de ciclu și $prec \leftarrow x$). Astfel, citirea se încheie în momentul în care $prec=0$ și $x=0$.

Folosim două variabile max și $maxg$, în care vom reține maximul fiecărui șir și maximul global. După citirea fiecărui nou x , dacă acesta este diferit de 0 actualizăm max (prin comparare cu max de la pasul anterior, unde inițial $max \leftarrow -MAXINT$), iar la întâlnirea valorii 0 actualizăm similar $maxg$ (prin comparare cu max).

► 49. Se citește un șir de numere întregi până când se introduce de două ori consecutiv aceeași valoare. Să se afișeze câte pătrate perfecte sunt în șir.

Exemplu: dacă șirul este 13 9 56 400 8 25 17 17, acesta conține trei pătrate perfecte (numerele 9, 400 și 25).

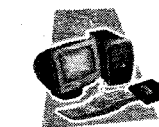
Indicații. Citim numerele pe rând într-un ciclu, în aceeași variabilă x . Pentru fiecare x citit, dacă este pătrat perfect ($\sqrt{x} = \lfloor \sqrt{x} \rfloor$) incrementăm un contor (inițializat cu 0). La fiecare pas, înainte de citirea unui nou număr în variabila x , salvăm x -ul curent într-o variabilă y . Astfel, ciclul se execută până când $x=y$.

► 50. Se numește număr "bine ordonat descrescător" un număr natural, cu proprietatea că cifrele sale citite de la stânga la dreapta sunt în ordine descrescătoare. Exemplu: 9653. Scrieți un algoritm care verifică dacă un număr natural x dat este "bine ordonat descrescător", afișând un mesaj.

Indicații. Presupunem mai întâi că numărul x este "bine ordonat crescător", inițializând cu $true$ o variabilă booleană. Reținem prima cifră a lui x ($x \bmod 10$) în variabila y . Apoi, într-un ciclu, extragem pe rând celelalte cifre, folosind algoritmul prezentat în acest capitol. La fiecare pas, dacă cifra curentă este mai mică decât cifra anterior extrasă y , variabila booleană devine $false$ (numărul nu mai poate fi bine ordonat crescător), apoi salvăm cifra curentă în y (cifra curentă va deveni cea precedentă la următorul pas).



Capitolul III: Elemente de bază ale limbajului C++



III.1. Noțiuni introductive



III.1.1. Structura unui program C++

Un exemplu de program. Funcția principală main

Vom începe cu un exemplu foarte simplu de program, pe care vom exemplifica structura generală a programelor scrise în limbajul C++:

```
#include <iostream.h>

void main ()
{
    cout << "Un exemplu ";
    cout << "foarte simplu";
}
```

Vom analiza pe rând toate elementele care apar în acest exemplu de program.

Ca orice limbă, și un limbaj de programare dispune de cuvinte proprii predefinite și are o sintaxă proprie. Cuvintele predefinite ale unui limbaj se numesc **cuvinte cheie**. În exemplul de mai sus întâlnim patru astfel de cuvinte cheie: `include`, `void`, `main`, `cout`. Limbajul C++ dispune de foarte multe astfel de cuvinte cheie, care vor face obiectul unei lecții viitoare.

După cum deja știți, algoritmul unui program este alcătuit dintr-una sau mai multe etape. În cadrul fiecărei etape se execută una sau mai multe **instrucțiuni**. Fiecare instrucțiune este "comandată" printr-o **linie de program**.

În general, un program C++ folosește niște **date**, care pot fi de două tipuri: **constante** și **variabile**. Constantele sunt date care nu-și modifică valoarea pe parcursul execuției programului. În schimb, o variabilă se caracterizează prin proprietatea că își poate modifica valoarea în orice moment. Fiecare variabilă are un nume sub care este "recunoscută" în cadrul programului, căruia îi spunem **identificator de variabilă**. Programul dat ca exemplu mai sus nu folosește nici un fel de date, singurul său efect fiind afișarea unui text pe ecranul monitorului, după cum vom vedea ceva mai târziu.

Instrucțiunile unui program C++ sunt grupate în niște entități numite funcții. Orice program trebuie să conțină cel puțin o funcție, cu numele **main**, numită **funcție principală**. Funcția principală **main** se execută automat atunci când lansăm în execuție programul.

Pe lângă funcția principală, un program poate să conțină oricâte alte funcții, cărora le vom spune generic "**funcții definite de către utilizator**". Fiecare astfel de funcție este "recunoscută" în program printr-un nume, pe care îl dăm noi (**identificator de funcție**).

Orice funcție C++, deci și funcția principală **main**, este alcătuită din două părți:

- "**antetul**" funcției;
- "**corpul**" funcției.

Când scriem o funcție, putem defini niște **parametri formali**. Aceștia sunt de fapt niște identificatori ale unor variabile pe care urmează să le folosească funcția atunci când se va executa. Pentru a comanda execuția unei funcții, trebuie să o apelăm. Dacă funcția are parametri formali, atunci la apel trebuie să-i dăm valori concrete ale acestora, valori numite **parametri actuali**. O funcție poate să returneze o anumită valoare, de un anumit tip. Dacă o funcție nu returnează nici o valoare, atunci vom spune că tipul valorii returnate este **void** ("nimic"). Prima linie a oricărei funcții, numită **antetul funcției**, trebuie să arate astfel:

<tipul_valorii_returnate> <numele_funcției> (<lista_parametrilor_formali>)

Nu trebuie să intrați în panică dacă nu ați înțeles în totalitate paragraful anterior! Deocamdată nu vom mai vorbi despre parametri unei funcții și despre tipul valorii returnate, dintr-un motiv foarte simplu: la nivelul clasei a IX-a vom scrie programe alcătuite numai din funcția principală **main**, în privința căreia lucrurile se prezintă foarte simplu. Deocamdată trebuie doar să rețineți doar că funcția principală **main** nu returnează nimic (tipul valorii returnate este **void**) și nu are parametri (lista parametrilor formali ... nu există !). Credem că este suficient pentru ca, pornind de la forma generală a antetului unei funcții dată mai sus, să deduceți cum arată antetul funcției principale **main**:

```
void main ()
```

Dacă nu există parametri se vor scrie numai parantezele, sau, eventual, cuvântul "**void**" între paranteze. De asemenea, dacă tipul valorii returnate este **void**, el poate să lipsească. Pornind de la aceste constatări, tragem concluzia că putem folosi și alte forme pentru antetul funcției **main**, și anume:

```
main ()
void main (void)
main (void)
```

Totalitatea instrucțiunilor unei funcții, adică totalitatea liniilor de program, alcătuiește "**corpul**" funcției. În mod obligatoriu, "corpul" oricărei funcții trebuie cuprins între "acoladă deschisă" și "acoladă închisă". O funcție poate să nu conțină nici o instrucțiune în "corpul" său (adică practic să nu execute nimic), dar și în acest caz acoladele trebuie să fie prezente. Avem convingerea că în programul dat ca exemplu mai înainte, ați identificat deja atât prezența acoladelor, cât și "corpul" funcției **main**:

```
cout << "Un exemplu ";
cout << "foarte simplu";
```

Așa cum am spus, programul de care ne ocupăm afișează un mesaj pe ecranul monitorului. Nu vom insista prea mult asupra instrucțiunilor de afișare, deoarece acestea vor face obiectul unei lecții separate. Totuși, pentru a înțelege "programul" nostru, vă vom dezvălui câteva "secrete":

- instrucțiunea de afișare este "**cout**" și trebuie urmată în mod obligatoriu de simbolul "<<" (de două ori caracterul "mai mic"); în program, ea este folosită de două ori;

- după simbolul "<<" urmează textul pe care trebuie să-l afișeze respectiva instrucțiune **cout**; un text este de fapt o *constantă șir de caractere*, adică un șir de caractere cuprins între ghilimele, care poate să conțină orice caractere (lucru pe care deja l-ați învățat, în primul capitol);

- orice instrucțiune de afișare se încheie cu caracterul "*punct și virgulă*";

Prima instrucțiune "**cout**" afișează textul "Un exemplu " (cu un spațiu la sfârșit), iar a doua instrucțiune tipărește textul "**de program**". Finalmente, cele două texte vor apărea "lipite" unul de altul, ca în imaginea următoare:

Un exemplu foarte simplu

Un set de două sau mai multe linii de program din corpul programului, se numește **secvență de program** sau **secvență de instrucțiuni**.

 **Observații:**

- Limbajul C++ face deosebire între literele mari și literele mici;
- Pe un rând de ecran putem scrie una sau mai multe instrucțiuni. În acest caz, instrucțiunile de pe același rând pot fi separate între ele prin oricâte spații.

În virtutea acestor observații, programul de mai sus poate fi scris și astfel:

```
#include <iostream.h>
void main ()
{
    cout << "Un exemplu ";   cout << "foarte simplu";
}
```

Directive preprocesor

Desigur că v-ați întrebat deja ce semnificație are prima linie a programului dat ca exemplu în lecția anterioară:

```
#include <iostream.h>
```

Acest aspect va fi explicat în continuare.

După ce am scris un program în limbajul C++, firește că dorim să-l executăm. Dar "creierul" calculatorului, adică microprocesorul, nu cunoaște decât un singur limbaj de programare, alcătuit numai din cifrele 0 și 1, numit "**limbajul cod-mașină**" (cifrele 0 și 1 ale bazei de numerație 2 se numesc **biți**, în interiorul microprocesorului toate informațiile și datele transmițându-se numai sub formă de biți). Prin urmare, este necesar ca programul scris de noi să fie transformat din limbajul C++ în limbajul cod-mașină pe care îl "înțelege" microprocesorul. Această operație se numește "**compilarea programului**". Ea se realizează printr-un program special, care face parte intrinsecă din limbaj, numit **compilator**. *Atunci când lansăm în execuție un program C++* (de exemplu prin apăsarea combinației de taste <Ctrl> și <F9>), *acesta este compilat automat* (putem însă "cere" ca programul să fie doar compilat, fără a se executa, apăsând tasta <F9> sau folosind opțiunea "Compile" din meniul limbajului). În urma compilării, calculatorul ne va semnala eventualele erori care împiedică executarea programului, și ne va transmite așa-numite "avertismente" ("warning"). Rezultatul compilării este ceea ce numim "**formatul direct executabil al programului**" (un fișier nume cu numele fișierului-sursă în care am scris programul, dar cu extensia ".exe").

Dar înainte de compilare, are loc o operație, numită **preprocesare**. Ea se realizează printr-un program special numit **preprocesor**. Preprocesorul este apelat automat înaintea compilării. În faza de preprocesare pot avea loc mai multe "acțiuni", dintre care deocamdată vom prezenta doar două: *includerea de fișiere externe în program și definirea de constante simbolice*.

Includerea de fișiere externe în program

Așa cum am spus, orice program C++ este alcătuit din funcții. În mod obligatoriu, într-un program C++ trebuie să existe funcția principală **main**, pe lângă care programul poate conține oricâte alte funcții scrise de noi, numite "funcții definite de utilizator". În momentul în care dorim să se execute o funcție definită de utilizator, trebuie să "comandăm" executarea ei. Vom spune că "**apelăm funcția**". O funcție definită de utilizator poate fi apelată atât din funcția principală **main**, cât și dintr-o altă funcție.

Dar orice mediu de programare în C++ dispune de un set important de **funcții predefinite**. Acestea au fost scrise de către autorii limbajului, fac parte intrinsecă din limbaj, și le putem folosi oriunde în cadrul unui program. Cu alte cuvinte, oricând dorim putem apela o astfel de funcție predefinită. Nu vom insista

prea mult deoarece și funcțiile predefinite fac obiectul unei lecții separate, dar pentru a înțelege despre ce este vorba vom da totuși un exemplu: pentru a "cere" calculatorului să calculeze radicalul dintr-un număr pozitiv, trebuie să apelăm funcția predefinită cu numele **sqrt**.

Toate funcțiile predefinite își au prototipul (definiția) în niște fișiere cu extensia ".h", numite **header-e**. Pentru a putea apela o funcție predefinită, trebuie ca la începutul programului să scriem o așa-numită "**directivă preprocesor**", cu ajutorul căreia includem în program header-ul în care se găsește prototipul funcției respective. Directivele preprocesor se plasează la începutul programului, ele se executându-se în faza de preprocesare. Sintaxa unei astfel de directive preprocesor este alcătuită din caracterul "#" ("diez"), cuvântul cheie "**include**", și numele header-ului scris între caracterele "<" și ">".

```
#include <nume_header>
```

Ca exemplu, revenim la funcția predefinită **sqrt** pentru calculul radicalului unui număr pozitiv. Această funcție își are prototipul în header-ul **math.h**. Pentru ca ea să fie "recunoscută", este necesar să includem în program header-ul în cauză, prin directiva preprocesor

```
#include <math.h>
```

Un header care nu trebuie să lipsească din nici un program, este **iostream.h**. Fără includerea acestuia nu se pot face nici un fel de citiri sau afișări de date. Acum sperăm că ați înțeles de ce programul dat în prima lecție începe cu linia

```
#include <iostream.h>
```

Alte header-e utilizate în mod uzual sunt: **conio.h**, **stdio.h** și **stdlib.h**. Despre acestea vom spune mai multe la momentul potrivit.

Dar nu numai header-e pot fi incluse în fișierul-sursă al unui program, ci și alte fișiere sursă. Într-o astfel de situație avem de-a face cu un program alcătuit din mai multe "module". Mai exact, într-un așa-numit "modul principal" vor fi înglobate alte "module de program". Includerea unui fișier-sursă se face tot cu directiva preprocesor "**#include**", cu deosebirea că numele fișierului se va scrie între "ghilimele", în loc de caracterele "<" și ">". De exemplu, dacă în programul nostru dorim să includem fișierul sursă cu numele **fis1.cpp**, vom scrie directiva

```
#include "fis1.cpp"
```

În forma de mai sus, calculatorul "subînțelege" că fișierul **fis1.cpp** se găsește în folder-ul (directorul) curent. Dacă fișierul cu pricina se află în alt folder / sub-folder, atunci la includere trebuie precizată "calea" prin care se poate ajunge la el. O astfel de "cale" este alcătuită dintr-un "lanț" de folder-e și sub-folder-e, separate între ele prin caracterele "\\" (de două ori caracterul "backslash").

Exemplu:

```
#include "C:\\Donaris\\Programe\\fis1.cpp"
```

• s-a inclus fișierul sursă `fis1.cpp`, aflat pe partiția "C", în folder-ul `Donaris`, subfolder-ul `Programe`.

III.1.2. Vocabularul limbajului și setul de caractere

Vocabularul limbajului

Vocabularul limbajului C++ este format din: *setul de caractere*, *identificatori*, *separatori* și *comentarii*.

Setul de caractere

Setul de caractere reprezintă totalitatea caracterelor cu ajutorul cărora se poate scrie un program C++.

• Setul de caractere este alcătuit din:

- ✓ literele mari și mici ale alfabetului englez: `a, b, ..., z, A, B, ..., Z`
- ✓ cifrele sistemului zecimal de numerație: `0, 1, ..., 9`
- ✓ caracterele speciale: `+ - * / = < > ( ) [ ] { } . , : ; " ' ~ ! @ # $ % ^ & _` etc.

III.1.3. Identificatori, separatori și comentarii

Identificatori

Așa cum s-a precizat anterior, prin **identificator** înțelegem un nume asociat unei constante, variabile, proceduri, funcții etc. Un identificator poate conține numai litere, cifre, și caracterul special `"_"` (liniuța de subliniere) și trebuie să înceapă obligatoriu cu o literă.

Exemple de identificatori corecți: `a2, b, val_medie`.

Exemple de identificatori incorecți: `1S` (începe cu o cifră), `m#t` (conține caracterul special `#`).

O categorie specială de identificatori este reprezentată de cuvintele cheie ale limbajului. Acestea au un înțeles bine definit și nu pot fi folosite ca identificatori ai unor constante sau variabile definite de utilizatori. Exemple de cuvinte cheie: `define, include, if, else, case, for, while, do, char, int, return, struct`, etc.

Separatori și comentarii

În cadrul unui program, o succesiune de caractere care luate împreună au o semnificație bine definită, se numește **unitate lexicală**. De exemplu, o unitate lexicală poate fi: un cuvânt cheie, un identificator, un caracter special, etc. Unitățile lexicale pot fi diferențiate între ele prin *separatori*.

Separatorii pot fi, după caz: `blanc` (caracterul "spațiu"), caracterul `" ; "` ("punct și virgulă") etc.

Dar un program trebuie să poată fi înțeles și de către alte persoane, nu numai de către autorul lui, și de aceea este util să conțină **comentarii**. Comentariile pot fi introduse în cadrul programului oriunde este legal să apară un caracter `"spațiu"` și sunt ignorate de către calculator la execuția programului.

Un comentariu poate fi scris în două moduri:

- ✓ între secvențele de caractere `"/* "` și `"*/ "`;
- ✓ punând secvența `"//"` în fața comentariului (în acest caz, compilatorul va considera drept comentariu tot ce urmează după secvența `"//"` până la sfârșitul rândului).

Facem precizarea că a doua formă este mai des întâlnită, motiv pentru care și noi o vom folosi mai frecvent pe parcursul acestei cărți

Exemplu:

```
#include <iostream.h>

void main()
{ /* începutul programului */
  // programul afișează două mesaje
  cout << "Un exemplu de program";
  cout << "O.K.";
} /* sfârșitul programului */
```

Instrucțiunea vidă

Exemplu:

Să urmărim programul următor:

```
#include <iostream.h>

void main()
{
;
}
```

Acest program nu face absolut nimic ! Corpul programului conține un singur caracter, și anume caracterul `"punct și virgulă"`. Despre acesta am spus că este un separator. În contextul de față, el joacă rolul a ceea ce numim

instrucțiunea vidă. O instrucțiune vidă nu are nici un efect, dar vom vedea în capitolul "Instrucțiunile limbajului" că uneori prezența acestuia într-un program este necesară.

Aplicație R.III.1. Program scris corect

Care dintre programele următoare sunt eronate ?

- | | | |
|--|--|---|
| a)
<pre>void main ()
{
;
}</pre> | b)
<pre>void main ()
{
;
};</pre> | c)
<pre>void main (void)
{
;
}</pre> |
| d)
<pre>void main ();
{
;
}</pre> | e)
<pre>main ()
{
;
}</pre> | |

Rezolvare

În toate variantele, corpul funcției `main` este corect. El conține o singură instrucțiune, și anume instrucțiunea vidă "punct și virgulă". Practic programele date nu fac nimic.

Numai în varianta d) este eronat antetul funcției `main`: nu se pune ";" la sfârșitul liniei de antet. Prezența caracterului ";" la sfârșitul funcției din varianta b), după acolada închisă, nu constituie eroare.

Răspuns corect: d).

Ați înțeles? Probleme cu răspuns scurt

►1. Pentru fiecare dintre afirmațiile de mai jos, precizați dacă este adevărată sau falsă.

- Într-un program, pe un rând de ecran putem scrie o singură instrucțiune.
- Sub sistemul de operare Windows, cuvântul "void" antetul funcției `main` poate să lipsească.
- Corpul funcției `main` trebuie întotdeauna cuprins între "{" și "}".

- a) A, F, A b) F, A, A c) F, A, F d) F, F, F e) A, A, F

►2. Care dintre identificatorii de mai jos sunt corecți și pot fi folosiți pentru variabile definite într-un program?

- a) 2a b) main c) aria_trapez d) x&2

►3. Pentru programul următor, alegeți răspunsul corect:

```
void main()  
{  
;  
}
```

- Programul este corect.
- Programul este greșit pentru că nu are nici o instrucțiune.
- Programul este greșit pentru că după "{" nu poate urma semnul ";".
- Programul este greșit pentru că nu are nici un header inclus.



III.2. Constante, variabile, operatori, expresii



III.2.1. Constante

Constantele sunt valori care nu se modifică pe parcursul execuției unui program. Ele se împart în două categorii: **constante numerice** și **constante nenumerice**. Constantele numerice sunt la rândul lor de două tipuri: **constante întregi** și **constante reale**. Din categoria constantelor nenumerice fac parte: **constantele caracter**, **constantele șir de caractere**, **constantele logice (booleene)** și **constantele simbolice**.

Tipurile de date întâlnite cel mai frecvent

Clasificarea și prezentarea detaliată a tuturor tipurilor de date ale limbajului C++ va face obiectul unui paragraf viitor. Amintim acum trei dintre acestea, pe care le vom folosi cel mai frecvent.

- **int** → cuprinde numerele întregi din intervalul $[-32767, 32767]$;
- **long** → permite memorarea unor numere întregi foarte mari.
- **float** → înglobează numerele reale (în formă normală sau exponențială).
- **char** → valorile de acest tip sunt caractere cuprinse între apostroafe.

Constante întregi

Sunt numere întregi alcătuite din cifrele zecimale de la 0 la 9, cu sau fără semn.

Exemplu: 32, -1628

Constante reale

În matematică, mulțimea numerelor reale include și numerele iraționale (de genul $\sqrt{2}$, $\sqrt{3}$ etc.), dar acestea nu pot fi reprezentate în calculator, din cauză

că au un număr infinit de zecimale. De aceea, constantele reale pot fi numai valori raționale, cu virgulă zecimală, dar care au un număr finit de zecimale. *Virgula zecimală se reprezintă prin caracterul "punct".*

Constantele reale se pot defini în două forme: *forma normală* și *forma exponențială*.

a) Forma normală.

Este cea pe care o cunoașteți deja. O constantă reală în formă normală este alcătuită din trei entități: *partea întreagă, punctul zecimal și partea zecimală*.

Exemplu: 2.34, -45.26, +2.34, -045.26, .56, -162.

Prezența caracterului "+" sau a cifrei 0 la începutul unei constante reale, deși inutilă, nu constituie eroare (lucru valabil și în cazul constantelor întregi). Partea zecimală a unei constante reale poate să lipsească, caz în care se subînțelege că aceasta este 0 (în exemplul anterior, scrierea "-162." este corectă, ea desemnând constanta reală -162.0). De asemenea, dacă partea întreagă a unei constante reale este 0, aceasta poate să lipsească (cazul scrierii ".56" din exemplul de mai sus, care desemnează constanta reală 0.56).

b) Forma exponențială (științifică).

În această formă se evidențiază o putere a lui 10 prin intermediul caracterului 'e' sau 'E'. Astfel, "a.bEc" sau "a.bec" este o scriere "științifică" a constantei $a.b \cdot 10^c$.

Exemplu: -2.3E-5 sau -2.3e-5 este echivalent cu $-2.3 \cdot 10^{-5}$

În funcție de numărul de octeți pe care îi ocupă în memorie, atât constantele întregi cât și cele reale pot fi împărțite în mai multe categorii (subtipuri). Despre acestea vom vorbi în lecția "Tipurile de date ale limbajului C++".

Constante caracter

Orice limbaj de programare folosește o codificare a caracterelor cu care lucrează. Limbajul C++ folosește standardul de codificare ASCII. Conform acestuia, fiecare caracter are un cod ASCII propriu, un număr întreg cuprins între 0 și 255.

Dispunerea caracterelor în setul ASCII respectă niște reguli. Astfel:

- Caracterele-litere mari 'A', 'B', ..., 'Z' au codurile cuprinse între 65 și 90;
- Caracterele-litere mici 'a', 'b', ..., 'z' sunt situate între codurile 97 și 122;
- Caracterele-cifre '0', '1', ..., '9' au codurile cuprinse între 48 și 57;
- Caracterele speciale sunt intercalate în tot intervalul [0, 255].

Trebuie observat un aspect foarte important: în setul ASCII cele două categorii de litere sunt diferențiate prin coduri distincte.

O constantă de tip caracter este un caracter ASCII cuprins între apostrofuri.

Exemplu: 'A', 'm', '&', '9' etc.

Observație:

A nu se confunda un caracter-cifă cu cifra zecimală; astfel, spre exemplu, caracterul '9' nu este totuna cu cifra 9!

Constante șir de caractere

Sunt șiruri de caractere cuprinse între ghilimele. O constantă șir de caractere poate conține orice caractere, inclusiv caractere speciale și caracterul spațiu.

Exemplu: "program", "Borland C++ 7.0", "AB#7", etc.

Constante întregi cu sens logic (boolean)

În cadrul unui algoritm putem întâlni condiții, care pot fi îndeplinite sau nu, și în funcție de care se vor decide acțiunile următoare. Orice condiție se caracterizează printr-o așa numită "valoare de adevăr" care ne spune dacă respectiva condiție este îndeplinită sau nu.

Pentru desemnarea celor două valori de adevăr posibile ale unei condiții, am folosit în capitolul anterior cuvintele **ADEVĂRAT** și **FALS**. Din păcate, limbajul C++ nu dispune de un tip de date pentru valori logice. De aceea, în C++ cele două valori de adevăr posibile sunt "simulate" prin intermediul numerelor întregi 0 și 1. Astfel:

- Numărul întreg 1 simulează valoarea de adevăr **ADEVĂRAT**
- Numărul întreg 0 simulează valoarea de adevăr **FALS**

Constante simbolice

Unei constante de orice tip i se poate atașa un *identificator de constantă*, adică un nume, definind astfel ceea ce numim **constantă simbolică**. În cadrul unui program, constantele simbolice se declară la început, cu ajutorul directivei-preprocesor "#define". Sintaxa este:

#define <nume> <valoare>

- ♦ <nume> → identificatorul de constantă simbolică pe care îl dăm constantei;
- ♦ <valoare> → valoarea constantei.

Exemplu:

Pentru a identifica numărul $\pi=3.1415927\dots$ pe care-l cunoașteți din geometrie, nu putem folosi litera grecească π , dar putem da acestei constante un nume sugestiv, ca de exemplu **PI**. Conform sintaxei de mai sus, declarația este

```
#define PI 3.14
```

Observație:

- Tipul unei constante (întreagă, reală, caracter, șir de caractere) rezultă chiar din valoarea sa.

În varianta Borland, limbajul C++ are câteva constante simbolice predefinite. De exemplu:

- Pentru desemnarea celei mai mari valori posibile de tip **int**, 32767, a fost atașată acestei valori o constantă simbolică cu identificatorul **MAXINT**;
- Constanta simbolică **MAXLONGINT=2147483647** identifică cea mai mare valoare posibilă de tipul **longint**;
- Numărul $\pi=3.1415927\dots$, pe care în exemplul anterior l-am asociat constantei simbolice **PI**, are o constantă simbolică predefinită, cu numele **M_PI**.

Facem o precizare foarte importantă: constantele simbolice predefinite se găsesc în header-e: **MAXINT** și **MAXLONGINT** sunt definite în header-ul **values.h**, iar **M_PI** în header-ul **math.h**. Firește că pentru a putea folosi aceste constante va trebui să includem în program header-ele respective, folosind directiva preprocesor **#include**:

```
#include <values.h>
#include <math.h>
```

Aplicație R.III.2. Corespondențe

Pentru fiecare dintre constantele aflate în coloana A), alegeți din coloana B) tipul său:

Coloana A)

- A1) 345
- A2) -2.
- A3) "57"
- A4) "include"
- A5) MAXINT
- A6) '9'
- A7) +72e-3

Coloana B)

- B1) constantă întreagă
- B2) constantă reală
- B3) constantă caracter
- B4) constantă simbolică predefinită
- B5) constantă șir de caractere

Rezolvare

345 este o constantă întreagă, iar -2. este o constantă reală cu partea zecimală nulă (de fapt numărul întreg -2 scris sub forma numărului real -2.0). Datorită ghilimelelor, "57" și "include" sunt constante șir de caractere (a nu se confunda șirul de caractere-cifra "57" cu numărul întreg 57, respectiv șirul "include" cu cuvântul cheie **include**). Apoi **MAXINT** este o constantă simbolică predefinită (cu valoarea 32767), iar '9' este o constantă caracter (atenție, nu cifra zecimală 9 !). În sfârșit, +72e-3 este constanta reală $72 \cdot 10^{-3}$ scrisă în formă exponențială. Prin urmare, corespondențele corecte sunt:

A1↔B1, A2↔B2, A3↔B5, A4↔B5, A5↔B4, A6↔B3, A7↔B2

III.2.2. Variabile

Reamintim:

- ❑ O variabilă este o entitate caracterizată prin trei atribute: *tip, valoare, adresă*, cu proprietatea că atributul valoare este modificabil. Cu alte cuvinte, o variabilă își poate modifica valoarea la fiecare nouă execuție a programului, sau în cursul aceleiași execuții.
- ❑ Tipul unei variabile definește mulțimea valorilor pe care le poate lua variabila respectivă. De exemplu, dacă o variabilă este de tipul **int**, atunci valoarea ei poate fi orice număr întreg cuprins între **-MAXINT** și **MAXINT**.
- ❑ La fel ca în matematică, o variabilă trebuie să aibă un identificator de variabilă, adică un nume.

Ca și constantele, variabilele trebuie declarate. În C++ declarațiile de variabile pot fi plasate oriunde în cadrul programului.

Sintaxă:

```
<tip> <v1>, <v2>, <v3>, ... ;
```

- ♦ <v1>, <v2>, <v3>, ... → identificatori de variabilă, separați prin "virgulă";
- ♦ <tip> → tipul variabilelor <v1>, <v2>, <v3>, ...

Orice variabilă va fi "văzută" și "recunoscută" de către compilator începând din locul unde a fost declarată și până la sfârșitul funcției în care se află declarația. Am definit astfel ceea ce numim "domeniul de vizibilitate al variabilelor".

Așa cum am spus, deocamdată vom scrie programe care conțin doar funcția principală **main**. Pentru ca variabilele să fie "văzute" în tot programul, le putem declara în două locuri:

– la începutul programului, imediat după directivele preprocesor și înaintea funcției `main`; variabilele astfel declarate se numesc **variabile globale**:

Exemplu:

```
#include <iostream.h>

int a2,x;           // declarăm variabilele a2 și x, ambele de tipul int
float media_info;   // declarăm variabila media_info de tipul float

void main ()
{
    .....
}
```

– la începutul funcției `main`, imediat după acolada deschisă care “inaugurează” corpul funcției; variabilele declarate în acest mod se numesc **variabile locale** funcției `main`; având însă în vedere că `main` este singura funcție din program, aceste variabile au același statut ca și cele globale, adică sunt “văzute” în toate liniile care urmează declarației.

Exemplu:

```
#include <iostream.h>

void main ()
{
    int a2,x;           // declarăm variabilele a2 și x, ambele de tipul int
    float media_info;   // declarăm variabila media_info de tipul float
    .....
}
```

La declararea unei variabile putem să o inițializăm, cu o valoare constantă, cu valoarea altei variabile, sau chiar cu valoarea unei expresii. Fiind vorba despre o variabilă, valoarea dată inițial nu este “bătută în cuie”, ea se poate modifica ulterior oricând pe parcursul programului.

Inițializarea unei variabile se face printr-o sintaxă de genul “<nume_variabilă>=<valoare_inițială>”. În cadrul unei linii de declarații, putem avea atât variabile inițializate, cât și variabile neinițializate.

Pentru început, ne vom obișnui să folosim numai variabile locale.

Exemplu:

```
int m=2;
// declarăm variabila m de tipul int, pe care o inițializăm cu valoarea 2.

float x, y=3.5, z=y, t=2*y-1, u;
// declarăm variabilele x, y, z și t, toate de tipul float;
// variabilele y și z au fost inițializate la declarare cu valoarea 3.5.
// variabila t a fost inițializată cu valoarea expresiei 2*y-1, adică 2*3.5-1=6.
```

Ceva mai târziu vom vedea că operatorul “=” cu ajutorul căruia inițializăm o variabilă, este de fapt operatorul de atribuire. Altfel spus, unei variabile `i` se atribuie la declarare o anumită valoare inițială. În exemplul de mai sus, variabilei `m` de tipul `int` `i` se atribuie valoarea inițială 2, variabila `y` de tipul `float` primește valoarea 3.5, iar variabila `z` primește valoarea lui `y`, adică tot 3.5.



III.2.3. Operatori

Sunt de mai multe tipuri:

- a) **operatori aritmetici**: + (adunare), – (scădere), * (înmulțire), / (împărțire de numere reale), % (restul împărțirii întregi);
- b) **operatori relaționali**: == (egal), != (diferit), <= (mai mic sau egal), >= (mai mare sau egal);
- c) **operatori logici**: && (“ȘI” logic), || (“SAU” logic), ! (“Negație” logică).
- d) **operatorul de atribuire** =

Folosirea operatorilor va fi ilustrată detaliat în exemplele concrete care urmează.



Ați înțeles ?

Probleme cu răspuns scurt

► 1. (Bacalaureat iulie 2008, varianta 62)

Care dintre următoarele afirmații este adevărată ?

- a) În limbajul C++ nu există noțiunea de operator relațional;
- b) +, –, *, /, sunt operatori relaționali;
- c) <, >, <=, >= sunt operatori relaționali;
- d) &&, ||, ! sunt operatori relaționali;

► 2. (Bacalaureat iulie 2009, varianta 24)

Care din următoarele afirmații despre operatorii din C++ este falsă?

- a) > este operator relațional;
- b) % este operator aritmetic;
- c) && este operator logic;
- d) <= este operator logic;

► 3. Pentru fiecare dintre afirmațiile de mai jos, completați cuvintele lipsă:

- a) Valoarea constantei simbolice `MAXINT` este ... și reprezintă
- b) Operatorul care realizează “SAU logic” între două valori de adevăr este ...
- c) Operatorul “<=” face parte din categoria operatorilor ...
- d) Constantele simbolice se declară cu cuvântul cheie ...

► 4. Care dintre următoarele valori sunt constante flotante scrise corect ?

- | | | | |
|------------|----------|------------|----------|
| 1) 2307.98 | 2) +54.3 | 3) -20.07 | 4) -198. |
| 5) .13 | 6) 1.9E4 | 7) -2.7E-3 | 8) 2.e+4 |

- a) 1), 2), 3), 6) și 7)
- b) Toate mai puțin 5)
- c) Toate
- d) Toate mai puțin 8)
- e) Primele cinci

►5. Definiți o constantă simbolică **PI** cu valoarea 3.14, folosind directiva-preprocesor **#define**.

- a) `#define 3.14 PI;` b) `#define PI 3.14;`
 c) `#define float PI 3.14;` d) `#define PI=3.14;`
 e) `#define float PI=3.14;`

►6. Care dintre următorii operatori este aritmetic?

- a) `\` b) `%` c) `!=` d) `||`

►7. Care dintre următoarele valori nu reprezintă constante șir de caractere?

- a) `"C++"` b) `'2853'` c) `"exemplu"` d) `"TRUE"`

►8. Care dintre valorile de mai jos *nu* sunt constante întregi scrise corect?

- a) 123 b) -17 c) --67 d) 0154

►9. Care dintre construcțiile de mai jos reprezintă constante caracter?

- 1) `" "` 2) `'\'` 3) `'a'` 4) `'''` 5) `\\'` 6) `'\13'` 7) `"a"` 8) `' '`
 a) 2), 3) și 8) b) toate c) toate mai puțin 5) și 6)
 d) 2), 3), 4) și 8) e) 3), 4) 5), 6) și 8)

►10. Care dintre următoarele declarații de variabile declară corect o variabilă **x** ce poate memora valori reale?

- a) `float x;` b) `double x;` c) `unsigned float x;` d) `x:float;`

►11. Care dintre liniile de program de mai jos realizează inițializarea corectă a variabilei **x** la declararea sa?

- a) `int x==2;` b) `x:int=2;` c) `int x=2;`
 d) `int x 2;` e) `x=2:int;`



III.2.4. Expresii



Reamintim:

- ❑ O **expresie** este alcătuită din *operandi* și *operatori*, și se caracterizează printr-un rezultat pe care-l numim *valoarea expresiei*.
- ❑ În esență, **operandii** reprezintă valorile care intră în calculul expresiei. Un operand poate fi o constantă sau o variabilă. **Operatorii** desemnează operațiile care se execută în cadrul expresiei. Ei au fost enumerați în paragraful anterior.
- ❑ În timpul execuției unui program, la întâlnirea unei expresii calculatorul evaluează expresia respectivă, astfel: se înlocuiesc variabilele cu valorile lor și se obține **valoarea expresiei**.

Un atribut important al oricărei expresii este *tipul* său. **Tipul unei expresii** reprezintă de fapt tipul valorii expresiei. De exemplu, dacă valoarea unei expresii este un număr întreg, atunci vom spune că expresia este de tip întreg.

Într-o expresie, ordinea în care se execută operațiile este dată de așa-numitele **reguli de prioritate a operatorilor**. Astfel, primele se execută operațiile aritmetice, apoi se evaluează operatorii logici, iar în final cei relaționali. Dintre operatorii aritmetici se execută mai întâi înmulțirile și împărțirile, apoi adunările și scăderile. Dintre operatorii logici, `!` (negația logică) are prioritate în fața celorlalți, apoi urmează `&&` ("ȘI" logic) și `||` ("SAU" logic) în această ordine. Operatorii de aceeași prioritate sunt evaluați în cadrul expresiei de la stânga la dreapta. Pentru a schimba această ordine implicită, vom folosi parantezele.

Dacă avem paranteze "îmbrycate", adică incluse una într-alta, atunci nu vom folosi parantezele pătrate și acolade, ci mai multe perechi de paranteze rotunde.

Expresiile se împart în două mari grupe: **expresii aritmetice** și **expresii logice**.

Expresii aritmetice

Sunt cele care efectuează operații aritmetice având ca rezultat un număr.

Exemplu:

Fie două variabile **x** și **y** de tipul **int** (valorile lor pot fi numere întregi).

• Expresia $\frac{x+y}{x-y}$ se scrie ca expresie C++ sub forma `(x+y)/(x-y)`. În absența parantezelor efectul ar fi fost altul. Scriind `x+y/x-y`, calculatorul ar "înțelege" și ar calcula $x + \frac{y}{x} - y$ (operatorul `/` are prioritate în fața adunării și scăderii).

Expresii logice

Ați învățat că o expresie logică descrie o condiție. Valoarea unei expresii logice reprezintă valoarea de adevăr a condiției aferente. O condiție poate fi adevărată sau falsă. Pentru desemnarea celor două valori de adevăr posibile, în pseudocod am folosit cuvintele cheie **"ADEVĂRAT"** respectiv **"FALS"**. Deoarece în limbajul C++ nu există tipul de date "logic", cele două valori de adevăr sunt simulate prin valorile întregi 1 respectiv 0. Mai exact, o expresie logică poate avea valoarea:

- 1, în cazul în care condiția pe care o definește este adevărată;
- 0, în caz contrar.

O expresie logică este construită de obicei cu ajutorul operatorilor relaționali. Aceștia sunt: `"=="` (operatorul de testare al egalității), `"<"` ("mai mic"), `">"` ("mai mare"), `"<="` ("mai mic sau egal"), `">="` ("mai mare sau egal"), `"!="` ("diferit").

Exemplu:

• Dacă a și b sunt variabile numerice (întregi sau reale), $a-10 \geq b$ este o expresie logică: pentru $a=12$ și $b=5$, expresia are valoarea 0, corespunzătoare valorii de adevăr "FALS" (" $12-10 \geq 5$? nu"); în schimb, pentru $a=18$ și $b=6$, expresia are valoarea 1, aferentă valorii de adevăr "ADEVĂRAT" (" $18-10 \geq 6$? da").

Mai multe expresii logice "elementare" pot fi compuse cu ajutorul operatorilor logici, obținându-se astfel o altă expresie logică. Operatorii logici au fost amintiți deja: "!" (negație logică), "&&" ("și logic"), "||" ("sau logic"). Aceștia desemnează operațiile logice pe care deja le cunoașteți din capitolul "Algoritmi. Principiile programării structurate", doar că acolo am folosit cuvintele "ȘI", "SAU". Expresiile logice "elementare" ce alcătuiesc expresia compusă se scriu între paranteze.

Cum se execută o astfel de expresie compusă? Se evaluează expresiile componente, iar între valorile lor (1 sau 0) se aplică operatorii logici.

Să vedem în continuare care va fi efectul aplicării tuturor operatorilor logici între valorile 1 și 0.

a) operatorul "!" realizează "negarea" valorii căreia se aplică. Astfel:

!1=0 (NE-ADEVĂRAT = FALS)

!0=1 (NE-FALS = ADEVĂRAT)

b) operatorul "&&" realizează "ȘI logic" între două valori booleene.

Rezultatul va fi ADEVĂRAT numai dacă ambii operanzi sunt ADEVĂRAT:

0 && 0=0 (FALS ȘI FALS = FALS)

0 && 1=0 (FALS ȘI ADEVĂRAT = FALS)

1 && 0=0 (ADEVĂRAT ȘI FALS = FALS)

1 && 1=1 (ADEVĂRAT ȘI ADEVĂRAT = ADEVĂRAT)

c) operatorul "||" realizează "SAU logic" între două valori booleene.

Rezultatul va fi ADEVĂRAT dacă cel puțin unul dintre operanzi este ADEVĂRAT:

0 || 0=0 (FALS SAU FALS = FALS)

0 || 1=1 (FALS SAU ADEVĂRAT = ADEVĂRAT)

1 || 0=1 (ADEVĂRAT SAU FALS = ADEVĂRAT)

1 || 1=1 (ADEVĂRAT SAU ADEVĂRAT = ADEVĂRAT)

Exemplu:

Fie expresia $(a > b \ \&\& \ b > c)$, în care variabilele au valorile $a=5$, $b=4$, $c=3$. Prin evaluare se obține " $(5 > 4 \ \&\& \ 4 > 3)$?" Dar " $5 > 4$ " are valoarea 1, " $4 > 3$ " are valoarea 1, iar " $1 \ \&\& \ 1$ " este 1.

Așadar valoarea expresiei $(a > b \ \&\& \ b > c)$ este 1.

Aplicație R.III.3. Expresie cu numere

Deduceți valoarea expresiei aritmetice

$$((12-2) * 2/4 - (11 - (2*5+11)/3)/2)/5$$

Rezolvare

Analizăm mai întâi expresia care se împarte la 5, și anume $(12-2) * 2/4 - (11 - (2*5+11)/3)/2$. Aici găsim o scădere, între termenii $(12-2) * 2/4$ și $(11 - (2*5+11)/3)/2$. Primul dintre acești termeni, $(12-2) * 2/4$, are valoarea $10 * 2/4$, adică $20/4$, adică 5 (reamintim că în cazul prezenței a doi operatori succesivi de același nivel de prioritate, ei se execută de la stânga la dreapta). Al doilea termen al scăderii, $(11 - (2*5+11)/3)/2$, înseamnă $(11 - 21/3)/2$, adică $(11 - 7)/2$, respectiv 2. Făcând în final diferența dintre cei doi termeni de mai sus și împărțind rezultatul la 5, obținem $(5-2)/5$, adică 0 (atenție, ambii operanzi ai împărțirii $3/5$ fiind întregi, limbajul C++ va furniza câtul întreg al împărțirii!).

Răspuns: 0

Aplicație R.III.4. Expresie aritmetică

Scrieți ca expresie C++, expresia matematică $\frac{x \cdot [(x+1) \cdot (y+1) - 2]}{x+y}$

Rezolvare

Este necesară folosirea parantezelor imbricate, care să înlocuiască parantezele pătrate și linia de fracție. Așadar:

– pentru a fi înmulțită cu x , expresia $(x+1) \cdot (y+1) - 2$ trebuie cuprinsă între paranteze rotunde; obținem astfel numărătorul $x \cdot ((x+1) \cdot (y+1) - 2)$.

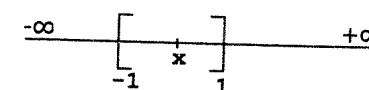
– linia de fracție este înlocuită prin operatorul "/", iar numărătorul și numitorul trebuie cuprins fiecare între paranteze; rezultă în final expresia C++:

$$(x \cdot ((x+1) \cdot (y+1) - 2)) / (x+y)$$

Aplicație R.III.5. Expresie logică

Scrieți ca expresie logică faptul că un număr x aparține intervalului $[-1, 1]$.

Rezolvare



Din desenul de mai sus se poate observa că dacă $x \in [-1, 1]$, atunci el este mai mare sau egal decât -1 și mai mic sau egal decât 1. Cu alte cuvinte, am întâlnit două expresii logice, " $x \geq -1$ ", " $x \leq 1$ " care trebuie să fie adevărate în același timp. Am format astfel expresia logică compusă:

$$(x \geq -1 \ \&\& \ x \leq 1)$$

- pentru $x=0$, valoarea expresiei este $1 \ \&\& \ 1$, adică 1 (" $0 \geq -1$ și $0 \leq 1$? da"), deci expresia are valoarea 1 .
- în schimb, pentru $x=2$, obținem $1 \ \&\& \ 0 = 0$ (" $2 \geq -1$ și $2 \leq 1$? nu", fiind adevărată numai prima dintre cele două expresii elementare componente).



Ați înțeles ?

Probleme cu răspuns scurt

► 1. (Bacalaureat iulie 2008, varianta 56)

Care dintre următoarele expresii reprezintă media aritmetică a patru numere reale memorate în variabilele a, b, c și d ?

- a) $(a+b+c+d) \cdot 0.25$; b) $((a+b)/2) + ((c+d)/2)/4$;
c) $a+b+c+d/4$; d) $(a+b+c+d) \cdot 0.4$;

► 2. (Bacalaureat iulie 2008, varianta 57)

Care dintre următoarele expresii logice este echivalentă cu condiția ca variabilele întregi a și b să aibă același semn și să fie nenule ?

- a) $a \cdot b > 0$;
b) $a + b > 0$;
c) $(a > 0 \ || \ b > 0) \ \&\& \ (a < 0 \ || \ b < 0)$;
d) $! (a < 0 \ \&\& \ b > 0) \ || \ (a > 0 \ \&\& \ b < 0)$;

► 3. (Bacalaureat iulie 2008, varianta 63)

Pentru variabilele întregi x, y, z și t ce memorează valorile $x=3, y=5, z=3, t=1$, precizați care dintre următoarele expresii logice are valoarea adevărată:

- a) $(t != 0) \ \&\& \ (x == z) \ || \ (y == z)$;
b) $(x > y) \ \&\& \ (t != 0)$;
c) $(x == z) \ \&\& \ (t == 0)$;
d) $! ((x != 0) \ \&\& \ (y != 0) \ \&\& \ (z != 0))$;

► 4. (Bacalaureat iulie 2008, varianta 35)

Stabiliți care dintre următoarele expresii este adevărată dacă și numai dacă numărul întreg x , nu aparține intervalului închis $[20, 100]$.

- a) $(x \leq 20) \ || \ (x \geq 100)$;
b) $(x < 20) \ || \ !(x > 100)$;
c) $(x \leq 19) \ || \ (x \geq 101)$;
d) $(x \leq 19) \ || \ (x \geq 101)$;

► 5. (Bacalaureat iulie 2008, varianta 29)

Care este valoarea expresiei: $(2 \cdot 3 - 5/2 - (3 - 7 \cdot 2))$?

- a) -2 b) 1.5 c) -1.5 d) 2

► 6. (Bacalaureat iulie 2008, varianta 42)

Precizați valoarea expresiei: $8/4/2 \cdot 2 \cdot 4 \cdot 8$.

► 7. (Bacalaureat iulie 2009, varianta 40)

Care este valoarea expresiei Pascal de mai jos?

$50 - (100 - 300/2/(2+3))$

- a) -30 b) 70 c) -20 d) 60

► 8. (Bacalaureat iulie 2009, varianta 22)

Care este valoarea expresiei C++ de mai jos dacă variabilele întregi a, b, c au valorile $a=20, b=3, c=5$?

$(a/b \ \% \ c \leq b) \ || \ !(b == a)$

► 9. a) Valoarea expresiei $(23 - (3+7)/2)/3 \cdot (15-11)$ este ...

b) Valoarea expresiei $9/(1/2-1) < 4 \cdot (3-8)$ este ...

► 10. (Bacalaureat, iulie 2001, varianta 1)

Care este valoarea expresiei $25 - (75 - 200/5/(2+3))$?

- a) -250 b) -58 c) -42 d) -10 e) 0 f) 30 g) 92 h) 108 i) 150

► 11. Fie variabilele întregi $a=1, b=2, c=3, d=4$. Care dintre construcțiile de mai jos sunt expresii scrise corect, cu valoarea 0 ?

- a) $!d \ || \ a$ b) $a+b>d$
c) $(a<b) != (b<c)$ d) $a=b<c$

► 12. Fie variabilele întregi a și b . Expresia C++ aferentă expresiei aritmetice

$$a \cdot b + \frac{a-b}{2}$$

este... ?

$$\frac{a-2}{b-2} - \frac{a}{b}$$

► 13. Știind că a, b, c, d, x sunt variabile reale cu $a \leq b$ și $c \leq d$, scrieți sub formă de expresie C++: $x \in [a, b]$ sau $x \in [c, d]$.

- a) $(x \geq a \ || \ x \leq b) \ \&\& \ (x \geq c \ || \ x \leq d)$
b) $!((x < a) \ \&\& \ !(x > b)) \ || \ !(x < c) \ \&\& \ !(x > d)$
c) $((x > a) \ || \ (x < b)) \ \&\& \ ((x > c) \ || \ (x < d))$
d) $(x > a \ \&\& \ x < b) \ || \ (x > c \ \&\& \ x < d)$

► 14. Care dintre următoarele expresii sunt adevărate dacă și numai dacă valorile variabilelor x și y sunt numere naturale consecutive?

- a) $(x-y==1) \ || \ (y-x==1)$ b) $(x==1) \ \&\& \ (y==2)$
c) $(x-y==1) \ \&\& \ (y-x==1)$ d) $y==x \pm 1$

► 15. Fie variabilele a, b, c de tipul `int`. Care dintre următoarele echivalențe între expresii sunt adevărate? (două expresii se consideră echivalente, dacă produc același rezultat în orice situație):

- a) Expresia $!(a!=b)$ este echivalentă cu $!a!=b$.
b) Expresia $a=(b<c)$ este echivalentă cu $(a=b)<c$.
c) Expresia $a \ \&\& \ b$ este echivalentă cu $a==0 \ \&\& \ b==0$.
d) Expresia $(a>c) \ \&\& \ (b>c)$ este echivalentă cu $a>c \ \&\& \ b>c$.

► 16. Fie două variabile întregi a și b , cu $a < b$. Cum se traduce matematic expresia $!(x < a) \ \&\& \ !(x > b)$?

- a) $x \notin (a, b)$ b) $x \in (a, b)$
c) $x \notin (-\infty, a) \cup (b, \infty)$ d) $x \in (-\infty, a) \cup (b, \infty)$

► 17. Se consideră variabilele întregi x , y și z , fiind cunoscute valorile $x=4$ și $y=2$. Care dintre expresiile de mai jos are valoarea 0?

- a) $x+y > x\&y+1$ b) $z=(x-y!=0)$
c) $!x$ d) $x-2*y=0$



III.2.5. Atribuirea în C++

Operatorul de atribuire "="

O variabilă poate primi valori, în corpul programului, în două moduri: prin atribuire și prin citire.

Prin atribuire, o variabilă primește o valoare concretă bine determinată. Operatorul de atribuire este "=".

Sintaxa:

`<nume_var> = <valoare>;`

- ♦ `<nume_var>` → numele variabilei (identificatorul de variabilă);
- ♦ `<valoare>` → valoarea care se atribuie variabilei respective;

Unei variabile i se poate atribui:

- o valoare constantă, dată direct sau prin intermediul unei constante simbolice;
- valoarea altei variabile stabilite anterior;
- valoarea unei expresii;

Exemplu:

```
#include <iostream.h>
#define a 3; // se definește constanta simbolică a, cu valoarea 3
void main ()
{
    int x=1,y,z,m; // variabilei x i se atribuie valoarea constantă 1, chiar la declarare
    float n;
    y=a;           // variabila y primește valoarea constantei simbolice a, adică 3
    z=y;           // variabila z ia valoarea variabilei y anterior atribuită, adică 3
    m=2*a+x-0.5; // variabila m ia valoarea expresiei 2*a+x-0.5; ⇒ m=2*3+1-0.5=6.5
    n=1.5;         // variabilei n i se atribuie constanta reală 1.5
}
```

La prima vedere, poate veți trage concluzia că valoarea unei variabile trebuie să fie exact de tipul variabilei. Într-adevăr, de exemplu, o variabilă de tipul întreg nu poate primi ca valoare un șir de caractere. Există însă și unele excepții de la această regulă, datorate așa-numitelor compatibilități între tipurile de date. De aceea, vom spune că tipul valorii unei variabile trebuie să coincidă sau să fie compatibil cu tipul variabilei. Astfel, de exemplu, o variabilă de un tip real (`float`, `double`, etc.) poate primi ca valoare un număr întreg, și invers, unei variabile de un tip întreg (`int`, `long`,

etc.) i se poate atribui un număr real. Faptul este perfect explicabil: așa după cum știți din matematică, mulțimea numerelor reale include în ea și mulțimea numerelor întregi, deci tipurile de date întregi și reale sunt compatibile între ele.

În C++ atribuirea se supune unei reguli generale de conversie, prin care valoarea care se atribuie este convertită implicit la tipul variabilei căreia i se atribuie. Unele conversii sunt însoțite de pierdere de informație, altele nu. Concret:

– un număr întreg se poate atribui unei variabile de un tip real, fără a avea loc vreo pierdere de informație; de fapt, întregul este privit ca un număr real cu partea zecimală nulă; de pildă, dacă atribuim numărul întreg 3 variabilei a de tipul `float`, atunci în realitate variabila a primește valoarea 3.0.

– invers însă, dacă atribuim o valoare reală unei variabile de un tip întreg, se va produce o pierdere de informație, prin trunchierea zecimalelor; de exemplu, dacă atribuim numărul real 2.65 variabilei b de tipul `int`, atunci în variabila b se va memora întregul 2.

Pentru a înțelege mai bine aspectele legate de compatibilitatea tipurilor, vă propunem să parcurgeți cu atenție exemplul care urmează.

Exemplu:

Câte dintre atribuirile programului următor sunt greșite?

```
#include <iostream.h>
void main ()
{
    int a,b,m;
    float n,p;
    a=5; b=4;
    n=a-b;
    m=1/a+a/b;
    a*b=p;
}
```

- a) nici una
b) una
c) două
d) trei

Primele două atribuirii sunt totalmente corecte: variabilele a și b primesc valorile 3 respectiv 4. Atribuirea `{n:=a-b;}` este la rândul-i corectă: `a-b` este un întreg (-1), care poate fi atribuit unei variabile reale, în speță n (de fapt numărul întreg -1 va fi interpretat ca numărul real -1.0).

A patra atribuire, prin care variabila m primește valoarea expresiei `1/a+1/b`, este și ea corectă. Valoarea expresiei `1/a+1/b` este `1/5+1/4`, adică 0.45. Valoarea expresiei `1/a+1/b` poate fi atribuită variabilei m de tip întreg, dar cu pierdere de informație, prin trunchierea părții zecimale. Astfel, valoarea variabilei m nu va fi 0.45, ci 0!

Cât despre ultima atribuire `a*b=p`, eroarea este evidentă: în stânga operatorului de atribuire "=" trebuie să se afle un identificator de variabilă iar în dreapta să o expresie, și nu invers.

Răspuns corect: b).

Atribuirea multiplă

În C++ putem atribui "dintr-o dată" aceeași valoare mai multor variabile, printr-o așa-numită "instrucțiune de atribuire multiplă", de forma:

```
<var_1>=<var_2>=<var_3>=...=<val>
```

- **<val>** reprezintă valoarea ce se atribuie variabilelor; ca de obicei, aceasta poate fi: o constantă, valoarea unei terțe variabile stabilită anterior, sau valoarea unei expresii;

- **<var_1>**, **<var_2>**, **<var_3>**, ... sunt identificatorii variabilelor.

Într-o astfel de expresie, atribuirile se efectuează succesiv de la dreapta la stânga. Astfel, mai întâi se atribuie valoarea **<val>** ultimei variabile din "lanțul" de atribuire, apoi penultima variabilă primește valoarea ultimei variabile stabilită anterior, ș.a.m.d.

Exemple:

```
• int x,y,z;  
  x=y=z=2;
```

Variabilele **x**, **y** și **z** primesc deodată aceeași valoare 2. Compilatorul va "rupe" instrucțiunea de atribuire multiplă în atribuirile elementare pe care le efectuează în următoarea ordine:

- atribuie valoarea 2 variabilei **z**;
- efectuează atribuirea **y=z**, prin care variabila **y** primește valoarea variabilei **z** stabilită anterior, adică tot 2;
- în sfârșit, face atribuirea **x=y**, în urma căreia și **x** va primi tot valoarea 2.

Deși în mod normal variabilele pot fi inițializate chiar la declarare, acest lucru nu este posibil dacă dorim o inițializare prin atribuire multiplă. Astfel, în loc de cele două instrucțiuni de mai sus, nu puteam scrie direct

```
int x=y=z=2;
```

O astfel de instrucțiune va produce eroare la compilare.

```
• int a=2,b=3,m,n,p,q;  
  m=n=p=q=2*a-b;
```

Variabilelor **q**, **p**, **n** și **m** li se atribuie, în această ordine, valoarea expresiei **2*a-b**, adică 1.

```
• int a=2,x,y,z;  
  x=(y=a-1)=z=a;
```

Această secvență ilustrează o tentativă de a schimba cu ajutorul parantezelor ordinea în care se efectuează atribuirile, care tentativă veți vedea însă că este zadarnică. Firește că parantezele obligă compilatorul să înceapă cu atribuirea **y=a-1** în urma căreia rezultă **y=1**, dar mai departe "lanțul" atribuirilor este parcurs sistematic de la dreapta la stânga, ignorându-se atribuirile deja efectuate grație parantezelor. Astfel:

- se efectuează cea mai din dreapta atribuire, **z=a**, în urma căreia rezultă **z=2**;

- urmează atribuirea **y=z**, deci variabila **y** va primi tot valoarea 2, anulându-se valoarea 1 pe care **y** a primit-o anterior (atribuirea **y=a-1** este acum ignorată, deoarece s-a făcut deja!);

- în fine, atribuirea **x=y** conduce la **x=2**.

Atribuirii cu sens logic

Unei variabile de tip întreg **i** se poate atribui valoarea de adevăr a unei expresii logice.

Reamintim:

O expresie logică conține operanzi de orice fel și operatori aritmetici, logici și relaționali. Ea descrie o condiție, care prin evaluare poate fi adevărată sau falsă, deci valoarea unei expresii logice poate fi: 1 dacă condiția pe care o definește este adevărată, sau 0 în caz contrar. Deoarece valoarea unei expresii logice se simulează printr-un număr întreg, firește că ea poate fi memorată prin atribuire într-o variabilă de tipul **int** sau **long**.

Exemplu:

Fie următoarele declarații de variabile:

```
int a=2,b=5,c=6,x,y;
```

Să analizăm următoarele expresii de atribuire:

```
x=(a<b && b<c);  
y=(a+b+c<=10);
```

- Variabilei **x** i se atribuie valoarea de adevăr a expresiei logice "**a<b && b<c**", care pe caz general poate fi 1 sau 0. Pentru valorile **a=2**, **b=5** și **c=6** date la declararea variabilelor, evaluarea expresiei logice are loc astfel: "**2<5 ȘI 5<6 ? da**", deci valoarea expresiei logice este 1. Prin urmare, variabilei **x** i se atribuie valoarea 1.

- Absolut similar, variabilei **y** i se atribuie valoarea expresiei "**a+b+c<=10**". În final vom avea **y=0** ("**2+5+6<=10 ? nu**").

Trebuie precizat faptul că parantezele de după operatorul de atribuire "=" nu sunt neapărat necesare. Grație priorității implicite a operatorilor, se execută mai întâi operațiile aritmetice și logice de după semnul "=", iar atribuirea este ultima. În exemplul de mai sus am pus totuși parantezele pentru a evidenția expresia logică ce se atribuie variabilei.

Conversii explicite

Să analizăm următoarea secvență de program:

Exemplu:

```
int x=4, y=13;  
float z;  
z=(x+y)/2;
```

În prima instrucțiune, se declară variabilele **x** și **y** de tipul întreg, inițializându-se totodată cu valorile 4 respectiv 13. A doua instrucțiune declară variabila **z** de tipul **float**,

iar a treia atribuie variabilei z valoarea expresiei $(x+y)/2$. Matematic, în urma evaluării acestei expresii, ar rezulta că valoarea ei este $(4+13):2$, adică $17:2$, adică 8.5 . Însă într-un program C++ lucrurile nu stau chiar așa, din cauza unei particularități a operatorului de împărțire $"/$. Atunci când deîmpărțitul și împărțitorul sunt numere întregi, operatorul $"/$ efectuează *împărțirea întreagă* a lor, adică rezultatul este *câtul întreg* al împărțirii. Cu alte cuvinte este suprimată partea zecimală a rezultatului. În cazul nostru, valoarea expresiei $(x+y)/2$ va fi 8 , reprezentând câtul întreg al împărțirii lui 17 la 2 . Atenție, acest lucru se întâmplă chiar dacă respectivul rezultat se atribuie unei variabile de tipul `float`! Așadar, în cazul de față, valoarea variabilei z nu va fi 8.5 , ci 8 .

Cum putem face să se efectueze împărțirea reală, astfel încât să se obțină și partea zecimală a rezultatului? Trebuie ca ori deîmpărțitul ori împărțitorul să devină de tip real. Astfel, se va împărți un număr întreg la un număr real sau un număr real la un număr întreg, caz în care compilatorul va efectua *împărțirea reală*, furnizând rezultatul complet, cu partea zecimală "intactă". Pentru aceasta, cel puțin unul din operanzi trebuie convertit către tipul `float`.

Conversia unui întreg la tipul `float` se poate face în două moduri:

a) Cu ajutorul operatorului `"punct"`; unui operand-constantă de tip întreg i se atașează caracterul "punct", care capătă rolul punctului zecimal, transformând întregul în numărul real corespunzător.

În cazul nostru, atribuirea se scrie sub forma

$z = (x+y) / 2.;$

Constantei întregi 2 i s-a atașat caracterul `"punct"`, transformând-o astfel în constanta reală 2.0 . În consecință, se va efectua $z = (4+13) / 2.0 = 8.5$. Acum împărțitorul este de tipul `float`, deci câtul împărțirii va reține și zecimalele. Valoarea lui z va fi în acest fel 8.5 .

Atenție, operatorul `"punct"` nu se poate aplica unui operator-variabilă. Astfel, este greșită o instrucțiune de genul

$z = (x+y.) / 2;$

b) folosind așa-numitul "operator de conversie explicită", aplicabil atât unui operand-constantă cât și unui operand-variabilă; înaintea operatorului ales pentru conversie, se va scrie cuvântul `"float"` între două paranteze rotunde.

Pentru expresia $(x+y)/2$ avem patru posibilități de a aplica acest tip de conversie:

• $z = ((float)x+y) / 2;$

S-a aplicat o conversie operandului x , scriind cuvântul `float` între două paranteze rotunde înaintea lui x , adică `(float)x`. Care va fi efectul? Valoarea întreagă 4 a lui x , este transformată în numărul real corespunzător, 4.0 . Rezultă $z = (4.0+7) / 2$. Numărătorul fiind de tipul `float`, se va obține $z = 8.5$.

• $z = (x+(float)y) / 2;$

Absolut similar, se convertește la flotant operandul y , rezultând $z = (4+7.0) / 2$, care este tot 8.5 .

• $z = (x+y) / (float)2;$

La fel ca o variabilă de tipul `int`, și o constantă întreagă (în cazul nostru 2) poate fi convertită la numărul real corespunzător folosind conversia explicită. Valoarea variabilei z va fi aceeași, 8.5 .

• $z = (float)(x+y) / 2;$

Cu ajutorul cuvântului `float`, se convertește valoarea expresiei $x+y$. Concret, se va calcula $x+y$ ca întreg, adică 17 , apoi această valoare este transformată în 17.0 , și în final se efectuează $17.0/2$. Se va obține tot $z = 8.5$.

Operatorii "+=", "-=", "*=", "/="

Dupa cum știți din pseudocod, dacă x este o variabilă de un tip numeric, atunci este corectă o atribuire de genul $x=x+2$, care se traduce astfel: "noua valoare a variabilei x va fi egală cu valoarea curentă a lui x plus 2 ", sau, mai simplist, "noul x va fi egal cu vechiul x plus 2 ".

În C++, atribuirea $x=x+2$ se poate scrie într-o formă mai elegantă: $x+=2$. În această construcție întâlnim operatorul `"+="` aplicat între doi operanzi, x și 2 . Primul operand reprezintă întotdeauna variabila a cărei valoare se mărește, iar al doilea operand precizează cu cât se mărește valoarea variabilei.

Pe caz general, sintaxa operatorului `"+="` este

$\langle var \rangle += \langle expr \rangle ;$

echivalent cu

$\langle var \rangle = \langle var \rangle + \langle expr \rangle ;$

- $\langle var \rangle$ reprezintă identificatorul unei variabile;
- $\langle expr \rangle$ poate fi o constantă, sau o altă variabilă sau o expresie.

Exemplu:

Fie variabilele x , y și z , cu valorile $x=4$, $y=5$, $z=7$;

• $x+=3;$

este echivalentă cu atribuirea $x=x+3$, adică mărește cu 3 valoarea variabilei x ; astfel vom avea $x=4+3=7$;

• $x+=y;$

echivalent cu $x=x+y$; în traducere, valoarea variabilei x se mărește cu valoarea variabilei y ; rezultă $x=7+5=12$;

• $x+=(2*z-y)/3;$

echivalent cu $x=x+(2*z-y)/3$, ceea ce înseamnă că la valoarea curentă a lui x se adaugă valoarea expresiei $(2*z-y)/3$, rezultând $x=12+(2*7-5)/3=15$.

Absolut similar funcționează operatorii `"-="`, `"*="`, și `"/="`:

➤ $\langle var \rangle -= \langle exp \rangle ; \rightarrow$ este echivalent cu $\langle var \rangle = \langle var \rangle - \langle exp \rangle ;$

➤ $\langle var \rangle *= \langle exp \rangle ; \rightarrow$ este echivalent cu $\langle var \rangle = \langle var \rangle * \langle exp \rangle ;$

➤ $\langle var \rangle /= \langle exp \rangle ; \rightarrow$ este echivalent cu $\langle var \rangle = \langle var \rangle / \langle exp \rangle ;$

De exemplu, $x*=2$ înseamnă $x=x*2$, iar $x/=2$ se traduce prin $x=x/2$.

Operatorii "++" și "--"

Dacă dorim să mărim cu 1 valoarea unei variabile oarecare x de tip numeric, vom folosi instrucțiunea de atribuire $x=x+1$, care, așa cum am arătat în lecția anterioară, poate fi scrisă și sub forma $x+=1$.

Pentru incrementarea cu o unitate a valorii unei variabile oarecare $\langle var \rangle$, există în limbajul C++ un alt operator special, și anume operatorul de incrementare "++". Acesta se poate aplica variabilei în două forme:

$\langle var \rangle ++$; → forma postfixată

$++\langle var \rangle$; → forma prefixată

Exemplu: pentru o variabilă x putem scrie instrucțiunile

$x++$;

$++x$;

Cu alte cuvinte, operatorul de incrementare "++" se atașează identificatorului de variabilă, fie înainte, fie după acesta. Așa cum sunt scrise instrucțiunile de mai sus, între cele două forme nu există nici o deosebire: ambele sunt echivalente cu atribuirea $x=x+1$.

Deosebirea dintre cele două forme apare atunci când operatorul "++" este folosit într-o expresie. Astfel:

- $\langle var \rangle ++$: mai întâi, calculatorul evaluează expresia, folosind valoarea curentă a lui $\langle var \rangle$, apoi incrementează $\langle var \rangle$ cu 1 ($\langle var \rangle = \langle var \rangle + 1$).
- $++\langle var \rangle$: mai întâi incrementează $\langle var \rangle$ cu 1, apoi evaluează expresia, folosind noua valoare a variabilei $\langle var \rangle$.

Mai facem mențiunea că, atunci când folosim operatorul de incrementare într-o expresie, nu sunt necesare parantezele, deoarece acest operator, fiind unar (se aplică unui singur operand), are prioritate maximă, înaintea tuturor operatorilor binari (aritmetici și logici) ce pot apărea în cadrul expresiei.

Absolut similar funcționează operatorul de decrementare "--", cu cele două forme ale sale, echivalente cu $\langle var \rangle = \langle var \rangle - 1$:

$\langle var \rangle --$; → forma postfixată

$--\langle var \rangle$; → forma prefixată

Exemplu:

Fie următoarea secvență de program:

```
int x=3,y=5,z,t;  
x++;  
++y;  
z=x-2*y++;  
t=--z;
```

• Instrucțiunea " $x++$ " este echivalentă cu " $x=x+1$ ", adică $x=x+1=4$.

• Incrementarea " $++y$ " are același efect, rezultând $y=y+1=5+1=6$.

• În expresia de atribuire " $z=x-2*y++$ ", operatorul de incrementare "++" se aplică variabilei y în formă postfixată. Evaluarea expresiei are loc astfel:

- ✓ Mai întâi se evaluează expresia folosind valoarea curentă a lui y , adică se evaluează expresia $z=x-2*y$ (pur și simplu facem abstracție de operatorul de incrementare "++"); rezultă $z=4-2*6=-8$;
- ✓ Apoi se incrementează y , mai precis se execută instrucțiunea " $y++$ ", echivalentă cu " $y=y+1$ "; prin urmare, vom avea $y=6+1=7$.

Fiind vorba despre folosirea operatorului într-o expresie, funcționarea acestuia este diferită față de situația în care l-am fi folosit în formă prefixată. Cu alte cuvinte, expresia dată nu este echivalentă cu " $z=x-2*++y$ "

• În sfârșit, expresia " $t=--z$ " conține decrementarea variabilei z aplicată în formă prefixată. Aceasta înseamnă că:

- ✓ Mai întâi se decrementează cu o unitate variabila z , ceea ce înseamnă $z=z-1=-8-1=-9$;
- ✓ Apoi are loc atribuirea $t=z$, în care variabilei t i se atribuie noua valoare a lui z rezultată în urma decrementării, respectiv $t=-9$.

Aplicație R.III.6. Atribuire

(Bacalaureat 2008, varianta 3)

Ce valori inițiale ale variabilelor x și y (ambele de tip întreg) trebuie puse în loc de "... " în secvența de program de mai jos, astfel încât la finele executării secvenței respectivele variabile să aibă valorile $x=2007$ și $y=2009$?

```
x=...;  
y=...;  
x=x-1;  
y=2*x+y;  
x=2*x+1;
```

Rezolvare

Știind valorile finale ale variabilelor x și y , vom pleca de la ultima atribuire. Aceasta poate fi citită sub forma "noul x " = $2 * \text{vechiul } x + 1$. Cum "noul x " este tocmai x -ul final, adică 2007, rezultă că "vechiul x " se poate obține din rezolvarea ecuației $2007 = 2 * x + 1$. Avem $2 * x = 2006$, deci x este 1003.

Între valoarea inițială a lui x (cea marcată prin "...") și valoarea din ultima atribuire (cea tocmai dedusă), mai există o instrucțiune care îl modifică pe x , și anume $x--$ adică $x=x-1$. Prin urmare, x -ul inițial este 1004, astfel încât, prin executarea expresiei aritmetice " $x-1$ ", variabila x să ajungă la valoarea 1003, cu care se intră în atribuirea următoare.

Aplicând și acestei atribuirii $y=2*x+y$ un raționament similar celui de mai sus, rezultă "noul y " = $2 * x + \text{vechiul } y$. Întrucât "noul y " este valoarea finală 2009, iar x

am dedus că este 1003, avem de rezolvat ecuația $2009 = 2 \cdot 1003 + y$, a cărei soluție este $y = 3$, care reprezintă tocmai y -ul inițial (pentru că valoarea lui y nu se mai modifică nicăieri în altă parte).

În concluzie, valorile inițiale ale celor două variabile, care trebuie puse în loc de "...", sunt $x = 1004$ și $y = 3$.

Ați înțeles ? Probleme cu răspuns scurt

► 1. (Bacalaureat iulie 2008, varianta 47)

Care dintre următoarele variabile nu-și modifică valoarea în urma executării secvenței de instrucțiuni indiferent de valorile inițiale ?

$c = 2 \cdot a + b$; $a = a + b$; $a = c - a$; $b = c - 2 \cdot a$;

- a) doar c b) doar a și b c) toate d) niciuna

► 2. (Bacalaureat iulie 2008, varianta 47)

Știind că variabila a și d sunt de tip `int`, iar variabilele b și c sunt de tip `float`, care dintre următoarele instrucțiuni de atribuire nu este corectă:

- a) $a = \text{sqrt}(a \cdot a)$; b) $c = 2 \cdot b \cdot a \cdot 2$; c) $d = b < c$; d) $b = b - c$;

► 3. (Bacalaureat iulie 2008, varianta 78)

Care dintre următoarele secvențe realizează interschimbarea valorilor variabilelor a și b (numere întregi de cel mult trei cifre) ?

- a) $a = b$; $b = a$; b) $a = \text{aux}$; $a = b$; $b = \text{aux}$;
c) $a = a + b$; $b = a - b$; $a = a - b$; d) $\text{aux} = a$; $b = \text{aux}$; $a = b$;

► 4. Se consideră următoarele declarații de variabile:

`int a, b; float c, d;`

Care din următoarele atribuiri sunt corecte:

- a) $a = a \cdot b$; b) $c = a + 1$; c) $b = +a + b$; d) $c \cdot d = a - b$;

► 5. (Bacalaureat iulie 2000, varianta 9)

Care dintre următoarele secvențe de instrucțiuni atribuie variabilei reale x media aritmetică a numerelor reprezentate de variabilele reale a , b și c ?

- a) $x = ((a + b) / 2 + c) / 2$; b) $x = a + b + c / 3$; c) $x = a / 3 + b / 3 + c / 3$; d) $x = (a + b + c) / 3$;

► 6. Care dintre instrucțiunile de atribuire de mai jos înjumătățește valoarea variabilei reale x ?

- a) $x += x / 2$; b) $x = x / 4 - 3 \cdot x / 4$;
c) $x = x / 4 + 3 \cdot x / 4$; d) $x -= x / 2$;

► 7. Fie atribuirea $\{x = (2 \cdot m \cdot n) / (m \cdot m - n \cdot n)\}$, în care m și n sunt variabile de tip întreg, iar x este o variabilă de tip real. Care dintre formulele de mai jos nu este echivalentă cu această atribuire ?

- a) $x = m / (m \cdot m - n \cdot n) \cdot (2 \cdot n)$; b) $x = m \cdot n / (m \cdot m / 2 - n \cdot n / 2)$;
c) $x = 2 \cdot m \cdot n / (m - n) \cdot (m + n)$; d) $x = 2 / (m / n - n / m)$;

► 8. Știind că formula de calcul pentru aria coroanei circulare este $\pi(R_1^2 - R_2^2)$, unde R_1 este variabila reală strict pozitivă reprezentând raza exterioară, iar R_2 este variabila reală pozitivă reprezentând raza interioară a coroanei, precizați care dintre următoarele instrucțiuni atribuie variabilei reale S valoarea ariei coroanei circulare (în variantele de răspuns s-a folosit constanta simbolică `M_PI`, predefinită în header-ul "math.h").

- a) $S = M_PI \cdot (\text{sqrt}(R_1) - \text{sqrt}(R_2))$; b) $S = M_PI \cdot (R_1 - R_2) \cdot (R_1 + R_2)$;
c) $S = M_PI \cdot R_1 \cdot R_1 - R_2 \cdot R_2$; d) $S = M_PI \cdot (R_1 \cdot R_1 - R_2 \cdot R_2)$.

► 9. Fie variabilele a și b de tip întreg, cu valorile $a = 3$ și $b = 0$. Care dintre instrucțiunile de mai jos atribuie valoarea 1 variabilei x de tip întreg ?

- a) $x = a \mid \mid b$; b) $x = (a > 0 \mid \mid b > 0)$
c) $x = !((a > 0 \ \&\& \ b <= 0))$ d) $x = (a == '3' \ \&\& \ b == '0')$

► 10. Fie variabilele x , y și u de tipul `int`. Care dintre instrucțiunile de mai jos mărește valoarea variabilei u cu câtul întreg al împărțirii lui x la y ?

- a) $u += x \cdot y$; b) $u = x \cdot y + u$;
c) $u = x / y$; d) $u += x / y$;

► 11. Pentru a atribui variabilei x rezultatul expresiei $\frac{2ab - c^2}{0.25}$, unde a , b , c desemnează variabile reale, vom scrie instrucțiunea de atribuire:

- a) $x = (2 \cdot a \cdot b) - (c \cdot c) / 0.25$; b) $x = 2 \cdot a \cdot b - c \cdot c / 0.25$;
c) $x = (2 \cdot a \cdot b) - (c \cdot c) \cdot 4$; d) $x = (2 \cdot a \cdot b - c \cdot c) \cdot 4$;



III.3. Citirea și afișarea datelor



III.3.1. Afișarea datelor fără formatare

În C++ există două tipuri de afișări ale unor date sau mesaje: afișarea fără formatare și afișarea cu formatare. În lecția de față vom vorbi despre afișarea fără formatare urmând ca afișarea cu formatare să fie tratată în cadrul secțiunii "Extindere" a prezentului capitol.

Afișarea unor texte (mesaje)

Acțiunile de afișare a unor date sau mesaje fără formatare se realizează cu ajutorul instrucțiunii "cout". Un exemplu de utilizare a instrucțiunii `cout` a fost prezentat deja în cadrul primei lecții a acestui capitol. Îl reluăm în continuare cu explicațiile necesare.

Exemplu:

```
#include <iostream.h>

void main ()
{
    cout << "Un exemplu ";
    cout << "foarte simplu";
}
```

Corpul funcției principale main este alcătuit din două instrucțiuni cout. Prima instrucțiune afișează textul "Un exemplu " (cu un spațiu la sfârșit), iar a doua instrucțiune va produce tipărirea textului "foarte simplu" (care va fi scris în continuarea primului mesaj). Per ansamblu, efectul execuției programului este apariția următorului text pe ecran:

```
Un exemplu_foarte simplu
```

Evident că același efect se putea obține și cu ajutorul unei sigure instrucțiuni cout, care să afișeze dintr-o dată întregul mesaj.

```
#include <iostream.h>

void main ()
{
    cout << "Un exemplu foarte simplu";
}
```

Să observăm sintaxa instrucțiunii cout. După cuvântul cout urmează simbolul "<<" (de două ori caracterul "mai mic"), apoi textul ce trebuie afișat, cuprins între ghilimele. Linia de program se încheie cu caracterul "punct și virgulă". Observați că textul în cauză nu este altceva decât o constantă șir de caractere, adică un șir de caractere cuprins între "ghilimele". Acesta nu are nici o semnificație pentru compilator, el va fi afișat ca atare, fără a fi "interpretat" în vreun fel.

Dar la un "apel" al instrucțiunii cout putem "comanda" afișarea mai multor texte (mesaje). Astfel, o variantă echivalentă a programului nostru este următoarea:

```
#include <iostream.h>

void main ()
{
    cout << "Un exemplu " << "foarte simplu";
}
```

Observați că o singură instrucțiune cout are "misiunea" de a afișa două mesaje. Așadar în C++ putem ca printr-o singură instrucțiune cout să afișăm mai mulți "itemi", dar într-o astfel de situație înaintea fiecărui "item" trebuie pus simbolul "<<".

În "programul" de mai sus, afișarea celui de al doilea text, "foarte simplu", începe în locul unde a ajuns cursorul pe ecran după tipărirea primului text ("Un exemplu "). Efectul va fi, la fel ca și în versiunile anterioare, apariția întregului mesaj "Un exemplu foarte simplu" pe un singur rând al ecranului. În cele ce urmează, vom vedea cum putem afișa un text pe mai multe rânduri de ecran.

Secvențe "Escape"

O secvență Escape este o succesiune de caractere cuprinse între ghilimele care nu se tipăresc, dar care au un anumit efect. Cea mai frecvent întâlnită secvență Escape este "\n", care are ca efect mutarea cursorului la începutul rândului următor pe ecran ("salt la rând nou"). Pentru a realiza doar această mutare, fără a se afișa nimic altceva, vom scrie instrucțiunea

```
cout << "\n";
```

Vom folosi o secvență Escape "\n" pentru a afișa mesajul nostru pe două rânduri de ecran, ca în "macheta" de mai jos.

```
Un exemplu_
foarte simplu
```

```
#include <iostream.h>

void main ()
{
    cout << "Un exemplu ";
    cout << "\n";
    cout << "foarte simplu";
}
```

Prima instrucțiune cout afișează textul "Un exemplu " pe rândul unde se află cursorul, a doua instrucțiune mută cursorul la începutul rândului următor fără a afișa nimic, iar a treia tipărește textul "foarte simplu" pe acest rând nou.

O secvență Escape "\n" poate fi atașată textului pe care dorim să-l afișăm. De exemplu, instrucțiunea

```
cout << "Un exemplu \n";
```

tipărește textul "Un exemplu " apoi mută cursorul la începutul rândului următor. Iată în continuare trei variante echivalente care se bazează pe această idee. Orice explicații suplimentare credem că sunt inutile.

```
#include <iostream.h>

void main ()
{
    cout << "Un exemplu \n";
    cout << "foarte simplu";
}
```

```
#include <iostream.h>
void main ()
{
    cout << "Un exemplu ";
    cout << "\nfoarte simplu";
}

#include <iostream.h>
void main ()
{
    cout << "Un exemplu \nfoarte simplu ";
}
```

Dar "\n" nu este singura secvență Escape a limbajului C++. Chiar dacă sunt mai puțin utilizate, le vom enumera și pe celelalte:

- \t → tabulator;
- \b → revenire cu o poziție înapoi pe rând (backspace);
- \r → mutarea cursorului în prima coloană de pe rândul curent;
- \f → salt de pagină.

Afișarea unor date de diverse tipuri

Cu ajutorul instrucțiunii `cout` se pot afișa nu numai mesaje, ci și date diverse: constante, de orice fel, valori ale unor variabile, valori ale unor expresii (inclusiv valoarea "de adevăr" a unei expresii logice), valori returnate de către funcții. Atunci când vrem să afișăm valoarea unei expresii, în cadrul instrucțiunii `cout` aceasta trebuie scrisă între paranteze rotunde.

Așa cum am spus, printr-o singură instrucțiune `cout` se pot afișa dintr-o dată mai mulți "itemi", cu precizarea că fiecare item trebuie să fie precedat de simbolul "<<".

Sintaxă:

```
cout << val_1 << val_2 << val_3...;
```

- ♦ `val_1, val_2, ...` sunt parametri care se vor afișa, precedați fiecare de succesiunea de caractere "<<".

Exemplu:

```
#include <iostream.h>
void main ()
{
    int x=16, y=4;
    cout << x << ' ' << y; // linia (L1)
    cout << "\nMedia aritmetica=" << ((x+y)/2.); // linia (L2)
    cout << "\n" << (x-2==y); // linia (L3)
}
```

- ♦ linia (L1) afișează: valoarea variabilei `x`, un caracter-spațiu și valoarea variabilei `y`;
- ♦ în linia (L2):
 - mai întâi se comandă saltul cursorului la începutul rândului următor pe ecran;
 - apoi se tipărește mesajul "media aritmetica=" (observați că în sursa programului primul item după `cout` este "\nMedia aritmetica=", ceea ce înseamnă că secvența Escape "\n" pentru saltul cursorului a fost "lipită" la începutul mesajului);
 - în sfârșit, se afișează valoarea expresiei $(x+y)/2$, care observați că este scrisă între paranteze rotunde.
- ♦ în linia (L3): cursorul este mutat din nou la început de rând prin intermediul secvenței Escape "\n", apoi se afișează valoarea de adevăr a expresiei logice " $x-2==y$ "; cum $16-2 \neq 4$, rezultă că se scrie 0.

Prin urmare, în urma execuției acestui program, ecranul va arăta ca în figură:

```
16 4
Media aritmetica=10
0
```

Oricât ar părea de ciudat, în instrucțiunea `cout` putem avea o atribuire. Mai exact, o linie de program de genul "`cout << (var=expresie)`" se execută astfel:

- variabilei `var` i se atribuie valoarea expresiei de după operatorul "=";
- se afișează valoarea variabilei `var` rezultată în urma atribuirii.

Menționăm însă că în instrucțiunea `cout`, atribuirea trebuie obligatoriu cuprinsă între paranteze rotunde.

Exemplu:

Fie secvența de program:

```
int a=3;
cout << (b=2*a+1);
```

Variabilei `b` i se atribuie valoarea expresiei aritmetice $2*a+1$, deci $b=2*3+1=7$; apoi se afișează valoarea lui `b`, adică 7.

Ați înțeles? Probleme cu răspuns scurt

► 1. (Bacalaureat iulie 2009, varianta 98)

Variabila întreagă `x` memorează o valoare mai mare ca 1000, formată doar din cifre distincte. Care dintre următoarele instrucțiuni C++ afișează o singură cifră?

- a) `cout<<x/1;` `printf("%d",x/1);`
- b) `cout<<x/100;` `printf("%d",x/100);`
- c) `cout<<x%100;` `printf("%d",x%100);`
- d) `cout<<x%10/1;` `printf("%d",x%10/1);`

► 2. (Bacalaureat iulie 2009, varianta 79)

Stabiliți ce se afișează în urma executării secvenței de instrucțiuni C++ alăturate dacă y este o variabilă reală iar x o variabilă întreagă.

```
y=10.1234;
x=(int)(y*100)/100;
printf("%d",x);
cout<<x;
```

a) 1012.34
b) 10.12
c) 0.12
d) 10

► 3. (Bacalaureat august 2001, varianta 1)

Ce valoare (în format exponențial) se afișează ca urmare a executării instrucțiunii: `cout << 48.0-(32/(2/2)+1)*;`

a) 0 b) 5 c) 8 d) 15 e) 17 f) 24 g) 32 h) 39 i) 41 j) altă valoare

► 4. Care dintre următoarele instrucțiuni afișează valoarea 2 dacă variabila de tip întreg n are valoarea 6?

a) `cout << (n/4)+1;` b) `cout << (n+1)/3;` c) `cout << (n-1)/2.;` d) toate

► 5. Pentru programul dat, precizați care dintre afirmațiile de mai jos sunt adevărate.

```
#include <iostream.h>
void main ()
{
    int a,c;
    float b;
    a=4; b=7;
    c=a*a-8;
    cout << (a<b && !(b>c)) << 1 << " "; // (1)
    cout << a << b << c; // (3)
}
```

- a) Atribuirea din linia (1) este greșită, deoarece variabila b este de tipul real iar valoarea atribuită 7 este un număr întreg.
b) Instrucțiunea de afișare din linia (2) este greșită deoarece parametrul "`a<b && !(b>c)`" nu identifică o valoare.
c) Instrucțiunea de afișare din linia (2) este greșită deoarece itemul "1" nu identifică o valoare.
d) În urma execuției programului se va afișa caracterul "spațiu".

11,478

► 6. Fie variabilele x de tip real și a de tip întreg. Care este formatul afișat în urma execuției instrucțiunilor de mai jos?

```
float x=32.5, a=6.0;
cout << a << x;
```

a) 6.032.5 b) 632.5 c) 6.00000032.5 d) 6.0032.50

► 7. Fie variabilele a , b și c de tipul întreg cu $a=3$, și instrucțiunile de mai jos. Care dintre aceste instrucțiuni *nu* afișează corect 345?

a) `cout << a << a+1 << a+2;` b) `cout << a << (b==a+1) << (c==a+2);`
c) `cout << a << a++ << ++a;` d) `cout << 100*a+10*(a+1)+(a+2);`

► 8. Fie variabilele x , y și z de tipul `int`, cu valorile $x=5$, $y=3$. Care dintre instrucțiunile de mai jos *nu* afișează valoarea 4?

a) `cout << x--;` b) `cout << ++y;`
c) `cout << 3*y/2;` d) `cout << (z=2*(x-y));`



III.3.2. Citirea datelor fără formatare

Ce se înțelege prin citirea datelor de la tastatură știți deja din capitolul anterior: pentru ca la fiecare nouă execuție a programului calculatorul să folosească alte valori ale variabilelor de intrare, este necesar ca aceste valori să fie citite. Citirea variabilelor se realizează cu instrucțiunea `cin`.

Sintaxă:

```
cin >> v1 >> v2 >> v3 ...;
```

$v_1, v_2, v_3, \dots$ sunt identificatorii variabilelor ale căror valori se citesc.

Observați că la un "apel" al instrucțiunii `cin`, pot fi citite "dintr-o dată" valorile mai multor variabile, dar în acest caz identificatorul fiecărei variabile trebuie precedat de succesiunea de caractere ">>".

Exemplu:

Prezentăm în continuare două variante ale unui program care calculează aria unui trapez cu dimensiunile b_1 , b_2 și h , reprezentând baza mare, baza mică și înălțimea.

// Programul trapez_1

```
#include <iostream.h>
void main ()
{
    float b1,b2,h,A;
    cout << "Dati valorile:\n"; // (L1)
    cin >> b1 >> b2 >> h; // (L2)
    A=(b1+b2)*h/2; // (L3)
    cout << "\n Aria=" << A; // (L4)
}
```

// Programul trapez_2

```
#include <iostream.h>
void main ()
{
    float b1,b2,h;
    cout << "\n b1=";
    cin >> b1;
    cout << " b2=";
    cin >> b2;
    cout << " h=";
    cin >> h;
    cout << "\n A=" << (b1+b2)*h/2;
}
```

În programul trapez_1:

- Linia (L1) comandă afișarea unui mesaj care va apărea înaintea citirii calculatorului va tipări textul **Dati valorile**
- Linia (L2) citește de la tastatură, printr-un singur "apel" al instrucțiunii `cout`, toate cele trei variabile de intrare. Calculatorul va aștepta să introducem pe rând trei numere, pe care le va memora în variabilele b_1 , b_2 și h .

- Instrucțiunea (L3) este o atribuire. Valoarea expresiei $(b1+b2)*h/2$ ce reprezintă aria trapezului, este atribuită variabilei **A**.
- Apoi, prin instrucțiunea (L4) se afișează textul **Aria=** urmat de valoarea variabilei **A**.

În programul **trapez_2** apar două modificări față de cealaltă variantă:

- Cele trei variabile se citesc separat, folosind pentru fiecare variabilă câte o instrucțiune **cin**, precedată de un afișarea unui mesaj cu **cout**;
- Valoarea ariei, adică valoarea expresiei $(b1+b2)*h/2$, nu se mai atribuie unei variabile **A**, ci se afișează direct. Cum? Expresia este scrisă direct în linia **cout**. Observați că ultima linie a programului afișează mesajul **A=**, urmat de valoarea expresiei $(b1+b2)*h/2$.

Aplicație R.III.7. Măsura unui unghi

Se citește de la tastatură măsura unui unghi **alfa**, exprimată în radiani. Știind că lui π radiani îi corespund 180° , să se afișeze măsura unghiului exprimată în grade, minute și secunde.

Reamintim: un grad = 60 minute, un minut = 60 secunde.

Rezolvare

Memorăm în variabilele **g**, **m**, **s** măsura unghiului **alfa** în grade, minute respectiv secunde. Pentru numărul π vom folosi constanta predefinită **M_PI**, cu valoarea 3.14159217..., existentă în header-ul **math.h**.

Pentru transformarea unghiului **alfa** din radiani în grade, folosim regula de trei simplă:

lui M_PI radiani		îi corespunde 180°
lui alfa radiani		îi corespunde g°

$$\Rightarrow g = (alfa * 180) / M_PI$$

Citim mai întâi de la tastatură măsura unghiului **alfa**, exprimată în radiani. Citirea variabilei **alfa** se face prin instrucțiunea **cin >> alfa;**, fiind precedată de afișarea mesajului **"Dati unghiul in radiani: "** (cu ajutorul instrucțiunii **cout**).

Apoi calculăm măsura unghiului în grăde: $g = (alfa * 180) / M_PI$. Deoarece un grad = 60 minute, rezultă că un unghi de măsură **g** grade are $g * 60$ minute, deci $m = g * 60$. Analog, măsura unghiului în secunde este $s = m * 60$, pentru că un minut = 60 secunde.

În final, afișăm cu **cout** măsura unghiului în grade, minute și secunde, respectiv valorile variabilelor **g**, **m**, **s**, între care intercalăm niște mesaje.

```
#include <iostream.h>
#include <math.h>

void main ()
{
    float alfa,g,m,s;
    cout << "\n Dati unghiul in radiani: ";
```

```
cin >> alfa;
g=(alfa*180)/M_PI;
m=g*60;
s=m*60;
cout << "\n" << M_PI;
cout << g << " grade " << m << " minute " << s << " secunde ";
}
```

Aplicație R.III.8. Interschimbarea a două numere

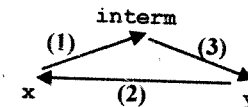
Scrieți un program care interschimbă între ele două numere reale citite de la tastatură, folosind "metoda paharelor".

Rezolvare

Denumirea de "metoda paharelor" provine de la o problemă practică: dacă vrem să schimbăm conținuturile a două pahare între ele fără să se amestece, unul din pahare trebuie vărsat temporar într-un al treilea pahar.

Judecând prin prisma acestui exemplu, nu putem să memorăm direct în **x** valoarea lui **y**, pentru că s-ar pierde valoarea inițială a lui **x**. Deci:

- > (1): valoarea lui **x** este salvată într-o variabilă "intermediară" ("temporară") **interm {interm=x};**
- > (2): în **x** se memorează valoarea lui **y {x=y};**
- > (3): în **y** se memorează valoarea pe care a avut-o inițial **x**, salvată în **interm {y=interm};**



Programul complet presupune citirea de la tastatură a celor două numere **x** și **y** (care se realizează cu instrucțiunea **cin** și este precedată de afișarea unui mesaj), apoi interschimbarea numerelor cu algoritmul de mai sus și în final afișarea valorilor interschimbate (cu **cout**).

```
#include <iostream.h>

void main ()
{
    int x,y,interm;
    cout << "\n Dati numerele: ";
    cin >> x >> y;
    interm=x;
    x=y;
    y=interm;
    cout << "\n Numerele interschimbate sunt " << x << " " << y;
```

Încercați singuri!

- > 1. Se citește de la tastatură măsura unui unghi **alfa**, exprimată în grade. Știind că măsurile de 180° îi corespunde π radiani, să se afișeze măsura unghiului în radiani.

►2. Scrieți un program care calculează și afișează volumul unui con, folosind formula $V = \frac{\pi \cdot R^2 \cdot h}{3}$. Valorile razei și înălțimii conului se citesc de la tastatură. Numărul π se va defini ca și constantă simbolică.

►3. Fiind date trei numere reale a, b, c reprezentând laturile unui triunghi oarecare, să se calculeze aria triunghiului, folosind formula lui Heron:

$$p = \frac{a + b + c}{2}, A = \sqrt{p \cdot (p - a) \cdot (p - b) \cdot (p - c)}.$$

►4. Realizați un program care citește de la tastatură două numere întregi, apoi afișează: media aritmetică, media geometrică, suma pătratelor și suma cuburilor celor două numere.

►5. Realizați un program care citește un număr de ani calendaristici, apoi tipărește numărul de luni, zile și ore cuprinse în acești ani. Se consideră că un an are 365 de zile.



III.4. Tipurile de date predefinite ale limbajului

Tipurile de date fundamentale (de bază) predefinite în limbajul C++, sunt: **tipurile întregi, tipurile reale, tipul character.**

Reamintim: tipul unei variabile desemnează mulțimea valorilor pe care le poate lua variabila și se precizează la declararea variabilei respective.



III.4.1. Tipurile numerice

Tipurile întregi

Până acum am vorbit despre tipul de date `int`, spunând că valorile de acest tip sunt numere întregi. În realitate, există mai multe tipuri de date întregi, care diferă prin intervalul valorilor posibile și prin numărul de octeți de memorie necesar. Acestea sunt:

- **Unsigned:** cuprinde numerele naturale din intervalul $[0, 65535]$, pe 2 octeți;
- **Unsigned int:** idem cu `unsigned`;
- **Int:** cuprinde numerele întregi din intervalul $[-MAXINT-1, MAXINT]$, pe 2 octeți (unde `MAXINT` este o constantă simbolică predefinită, existentă în header-ul `values.h`, cu valoarea 32767);

- **Short:** idem cu `int`;
- **Long:** cuprinde numerele întregi din intervalul $[-MAXLONG, MAXLONG]$, pe 4 octeți (unde `MAXLONG` este 2^{31} , adică 2147483647);
- **Unsigned long:** cuprinde numerele întregi din intervalul $[0, 2 \cdot MAXLONG]$, pe 4 octeți;

Exemplu:

Pentru a declara o variabilă x care să memoreze numere întregi foarte mari vom scrie:

```
var x:longint;
```

Tipurile reale

Din această categorie, până acum am folosit tipul `float`. A venit momentul să enumerăm toate tipurile reale:

- **Float:** numere reale din intervalul $\pm[3.4 \cdot 10^{-38}, 3.4 \cdot 10^{38}]$, reprezentate pe 4 octeți;
- **Double:** numere reale din intervalul $\pm[1.7 \cdot 10^{-308}, 1.7 \cdot 10^{308}]$, reprezentate pe 8 octeți;
- **Long double:** numere reale din intervalul $\pm[3.4 \cdot 10^{-4932}, 1.1 \cdot 10^{4932}]$, reprezentate pe 10 octeți;



III.4.2. Împărțirea cu rest. Operatorul "%"

Asupra valorilor care aparțin unuia din tipurile numerice (întregi sau reale) se pot aplica operatorii aritmetici pe care-i cunoașteți deja "+", "-", "*", "/".

După cum ați învățat deja, operatorul de împărțire "/" are o particularitate: dacă deîmpărțitul și împărțitorul, sunt ambele numere întregi, atunci se efectuează **împărțirea întreagă**, adică rezultatul va fi **câtul împărțirii întregi** (practic, este suprimată partea zecimală a câtului). Atunci când vorbim despre împărțirea întreagă, firește că trebuie să avem învedere nu numai câtul, ci și **restul împărțirii**. Pentru obținerea restului există operatorul "%". De exemplu, prin împărțirea întreagă a lui 32 la 5, se obține câtul întreg 6 și restul 2, și, firește, $27 = 6 \cdot 5 + 2$ ("deîmpărțitul = câtul * împărțitorul + restul"). Pe caz general:

Dacă operandii x și y sunt ambii numere întregi, atunci:

- ✓ $x / y \rightarrow$ reprezintă câtul întreg al împărțirii lui x la y ;
- ✓ $x \% y \rightarrow$ reprezintă restul împărțirii întregi a lui x la y .

Exemplu:

```
#include <iostream.h>
int x,y;
int c1,c2,r1,r2;
```

```

void main()
{
    x=14; y=5;
    c1=x / y;           {c1=14 / 5=2, câtul împărțirii întregi a lui 14 la 5}
    r1=x % y;           {r1=14 % 5=4, restul împărțirii întregi a lui 14 la 5}
    cout << c1 << ' ' << r1;    {afișează c1 și r1 separate printr-un spațiu}
}

```

În ceea ce privește restul împărțirii, o atenție mai specială trebuie acordată cazului în care cel puțin unul dintre operandii împărțirii este negativ. Astfel, de exemplu:

$-14 \% 3 = -2$, $14 \% -3 = 2$ și $-14 \% -3 = -2$

Regula care se aplică este următoarea: *se calculează câtul împărțirii operandilor fără semn, apoi rezultatul primește semnul primului operand.*

Pentru a evalua rezultatul se poate folosi și următorul adevăr matematic:

$D = C \cdot \hat{I} + R$ (unde D =deîmpărțitul, $\hat{I}$ =împărțitorul, C =câtul, R =restul), care se mai poate scrie $R = D - C \cdot \hat{I}$, adică $a \% b = a - (a / b) * b$.

Din considerentele de mai sus, se observă că operatorul `mod` poate fi folosit pentru a testa divizibilitatea a două numere întregi. Astfel:

- ✓ x este divizibil cu y dacă și numai dacă $x \% y == 0$ (de exemplu, 15 nu este divizibil cu 3 deoarece $15 \% 3$ este egal cu 0);
- ✓ x nu este divizibil cu $y \Leftrightarrow x \% y != 0$ (de exemplu, 17 nu este divizibil cu 3 deoarece $17 \% 3$ este diferit de 0).

Operatorul `mod` are prioritatea cea mai mare, chiar și înaintea operatorilor de adunare și scădere. Într-o expresie care conține mai mulți operatori `"/` și `"%`, aceștia se evaluează de la stânga la dreapta. Astfel:

- ✓ restul împărțirii întregi a lui $a+b$ la c se scrie $(a+b) \% c$, care nu e totuna cu $a+b \% c$;
- ✓ expresia $a / b \% c$ este evaluată sub forma $(a / b) \% c$.

Aplicație R.III.9. Expresii cu / și %

Valoarea variabilei x este un număr natural. Reprezentarea în baza 10 a lui x este alcătuită din exact trei cifre, dacă și numai dacă:

- a) $x/100 == 0 \ \&\& \ x \% 1000 == 0$
- b) $x/1000 > 0 \ \&\& \ x/100 == 0$
- c) $x \% 100 > 0 \ \&\& \ x/1000 == 0$
- d) $x/1000 == 0 \ \&\& \ x/100 > 0$

Rezolvare

Din start, varianta a) cade, întrucât condiția " $x/100 == 0$ " nu e îndeplinită (câtul împărțirii unui număr x de trei cifre la 100 este o valoare alcătuită dintr-o singură cifră; de exemplu, 738 împărțit la 100 dă câtul 7 și restul 38). Reamintim faptul că, fiind vorba de două expresii logice elementare legate prin operatorul "`&&`", dacă prima dintre respectivele expresii este falsă, și condiția în ansamblu are valoarea "`false`". De asemenea și varianta b) se elimină, deoarece câtul împărțirii unui număr x de trei cifre la 1000 este 0, deci condiția " $x/1000 > 0$ " este falsă.

Fără prea mare efort, putem observa că expresiile logice din variantele c) și d) sunt îndeplinite de către orice număr de trei cifre, deci aparent ambele variante ar fi corecte. Numai că, atunci când în enunțul problemei apar două condiții legate între ele prin sintagma "*dacă și numai dacă*", trebuie să vedem dacă este asigurată valabilitatea stării de "adevărat" în ambele sensuri. Dar în cazul expresiei din varianta c), reciproca nu mai este adevărată. De ce? Enunțul reciproc ar fi ceva de genul "dacă expresia de la c) este adevărată, atunci numărul x are exact trei cifre". Dar, pe lângă numerele de trei cifre, mai există și alte tipuri de numere care îndeplinesc condiția " $x \% 100 > 0 \ \&\& \ x/1000 == 0$ "! Bunăoară, pentru $x=23$, avem $23 \% 100 > 0$ (restul împărțirii lui 23 la 100 este 23), și $23/1000 == 0$ (câtul împărțirii lui 23 la 1000 este într-adevăr zero). Practic, toate numerele de două cifre îndeplinesc la rândul lor condiția de la c), deci e clar că respectiva condiție nu asigură limitarea valabilității strict la domeniul numerelor de trei cifre! În schimb, acest lucru se întâmplă în cazul expresiei din varianta d), care este cea corectă. Într-adevăr, orice valoare de trei cifre îndeplinește condiția " $x/1000 == 0 \ \&\& \ x/100 > 0$ ", și în același timp nici o valoare care are un alt număr de cifre nu o îndeplinește.

Răspuns corect: d)

Ați înțeles?

Probleme cu răspuns scurt

► 1. (Bacalaureat iulie 2008, varianta 3)

Care dintre următoarele expresii are valoarea `TRUE` dacă și numai dacă numărul natural c este multiplu comun al numerelor naturale nenule a și b .

- a) $c \% a == 0 \ \&\& \ c \% b == 0$;
- b) $a \% c == 0 \ \&\& \ b \% c == 0$;
- c) $a \% b == 0 \ \&\& \ c \% b == 0$;
- d) $c \% a == 0 \ \&\& \ c \% b == 0$;

► 2. (Bacalaureat iulie 2008, varianta 28)

Determinați cea mai mică valoare pozitivă pe care o poate primi variabila întreagă x , astfel încât expresia $(x/2) * 2$ să fie egală cu x .

► 3. (Bacalaureat iulie 2008, varianta 16)

Care dintre următoarele instrucțiuni inserează o cifră c în fața ultimei cifre a numărului natural n , memorând rezultatul tot în n ?

- a) $n := (n \% 10 * 10 + c) * 10 + n / 10$;
- b) $n := n / 10 + c + n \% 10$;
- c) $n := (n / 10 * 10 + c) * 10 + n \% 10$;
- d) $n := (n / 10 + c) * 10 + n \% 10$;

► 4. (Bacalaureat iulie 2008, varianta 76)

Variabila întreagă nr trebuie să rețină câți multipli mai mari decât 0 și mai mici sau egali decât n are numărul k (n și k sunt numere naturale date). Care este expresia cu care trebuie completată atribuirea $nr = \dots$?

- a) $n \& k$ b) $n - n/k$ c) n/k d) $n - n \& k$

► 5. (Bacalaureat iulie 2008, varianta 57)

Dacă n este un număr natural de exact două cifre $n = ab$, definim răsturnatul lui n ca fiind numărul ba dacă $b \neq 0$ și respectiv a dacă $b = 0$. De exemplu, răsturnatul lui 12 este 21, iar răsturnatul lui 10 este 1. Care dintre următoarele expresii reprezintă răsturnatul unui număr natural n cu exact două cifre?

- a) $10 * n \& 10 + n \& 10$; b) $n / 10 * 10 + n \& 10$;
c) $10 * n / 10 + n / 10$; d) $n \& 10 * 10 + n / 10$;

► 6. (Bacalaureat iulie 2008, varianta 69)

Știind că variabilele a , b și c sunt de tip întreg, care este condiția ca numărul natural memorat de variabila c să fie un multiplu comun al numerelor naturale memorate de variabilele a și b ?

- a) $c \% a * c \% b == 0$; b) $a \% c + b \% c == 0$;
c) $c \% a + c \% b == 0$; d) $(a \% c == 0) \&\& (c \% b == 0)$;

► 7. (Bacalaureat iulie 2009, varianta 53)

Fie x o variabilă care memorează un număr natural. Care dintre următoarele expresii $c++$ are valoarea 0 dacă și numai dacă numărul memorat de x NU este divizibil cu 3?

- a) $x \% 3$; b) $(1 - x \% 3) + (2 - x \% 3)$;
c) $(1 - x \% 3) * (2 - x \% 3)$; d) $3 - x \% 3$;

► 8. (Bacalaureat iulie 2000, variantele 1 și 6)

Care dintre următoarele expresii sunt adevărate dacă și numai dacă numărul întreg x este impar negativ?

- a) $(x \% 2 == 1) \&\& (x < 0)$ b) $x \% 2 != 0 \&\& x < 0$
c) $!(x \% 2 == 0 \&\& x >= 0)$ d) $!(x \% 2 == 0 \&\& x >= 0)$
e) $x \% 2 = 1 \&\& x < 0$

► 9. (Bacalaureat, august 2000, varianta 3)

Care dintre următoarele instrucțiuni mărește valoarea variabilei reale u cu jumătate din valoarea variabilei reale x ?

- a) $u += x \% 2$; b) $u = x + u / 2$;
c) $u -= x / 2$; d) $u = x / 2 + u$;

► 10. (Bacalaureat iulie 2000, varianta 3)

Care dintre următoarele expresii sunt adevărate dacă și numai dacă valorile variabilelor întregi x și y sunt numere pare?

- a) $x - y == 2$ b) $x * y \% 4 == 0$
c) $(x + y) \% 2 != 0$ d) $x \% 2 == 0 \&\& y \% 2 == 0$

► 11. Fiind date două numere întregi x și y , scrieți expresia care descrie condiția ca partea întreagă din media aritmetică a lui x și y să fie divizibilă cu 3.

- a) $(x + y) / 2 \% 3 == 0$ b) $(x + y) / 2 \% 3 == 0$
c) $x + y / 2 \% 3 == 0$ d) $((x + y) \% 2) / 3 == 0$

► 12. (Bacalaureat iulie 2003, varianta B)

Variabila întreagă p trebuie să rețină numărul de numere naturale mai mari sau egale cu 0 sau mai mici sau egale cu n care sunt divizibile cu k . Cu ce expresie trebuie completată atribuirea " $p = \dots$ "?

- a) $n \% k + n / k$ b) $n \% k + 1$ c) n / k d) $n / k + 1$

► 13. (Bacalaureat iulie 2003, varianta B)

Știind că n și p sunt variabile întregi ce memorează numere naturale de cel mult trei cifre, condiția ca n și p să fie de aceeași paritate (ambele pare sau ambele impare) este:

- a) $(n + p) \% 2 != 0$ b) $(n * p) \% 2 != 0$ c) $(n * p) \% 2 == 0$ d) $(n + p) \% 2 == 0$

► 14. (Bacalaureat iulie 2002, varianta 3)

Fie n un număr natural de trei cifre reprezentat în baza 10. Care dintre următoarele instrucțiuni atribuie variabilei întregi s valoarea cifrei sutelor a lui n ?

- a) $s = n \% 10$; b) $s = n / 10$;
c) $s = (n - n \% 10) / 10$; d) $s = (n - n \% 100) / 100$



III.4.3. Funcții predefinite pentru tipurile numerice

Noțiunea de funcție a fost definită deja. Reamintim:

O funcție predefinită este un bloc de instrucțiuni scris de către autorii limbajului și care face parte intrinsecă din limbaj. Pentru a "comanda" execuția unei funcții, trebuie să o apelăm. O funcție poate avea unul sau mai mulți parametri, ce definesc valorile cu care lucrează în timpul execuției sale. La apel, vom transmite procedurii valori pentru parametrii care au fost definiți. În urma execuției cauzată de un apel, funcția returnează (întoarce) o anumită valoare. Pentru constantele și variabilele de tipuri reale, există următoarele funcții predefinite:

- **abs(x)** returnează modulul numărului x ;
- **sqr(x)** returnează x^2 ;
- **sqrt(x)** returnează $\sqrt{x}$;
- **sin(x)** → returnează sinusul unghiului x (exprimat în radiani);
- **cos(x)** → returnează cosinusul unghiului x ;
- **tan(x)** → returnează tangenta unghiului x ;
- **asin(x)** → returnează "arc-sinus" din x (adică unghiul al cărui sinus este x);
- **acos(x)** → returnează "arc-cosinus" din x (adică unghiul al cărui cosinus este x);
- **atan(x)** → returnează "arc-tangenta" din x (unghiul al cărei tangentă este x);
- **pow(x, y)** → returnează valoarea lui x^y (" x la puterea y ");

- **ceil(x)** → rotunjește pe x la cel mai apropiat întreg mai mare decât x și returnează rezultatul

Exemple: `ceil(4.2)=5;` `ceil(4.7)=5;`
 `ceil(-4.2)=-4;` `ceil(-4.7)=-4.`

- **floor(x)** → trunchiază pe x la cel mai apropiat număr întreg mai mic decât el.

Exemple: `floor(4.2)=4;` `floor(4.7)=4`
 `floor(-4.2)=-5;` `floor(-4.7)=-5.`

Valoarea returnată de către o funcție poate fi afișată, atribuită unei variabile, folosită într-o expresie, transmisă ca parametru altei funcții.

Exemple:

Considerăm variabilele reale x, y, z și t , cu $x=4$. Analizăm câteva instrucțiuni:

<code>cout << sqrt(x);</code>	Afișează valoarea returnată de către funcția <code>sqrt(x)</code> , adică $\sqrt{x}$, deci tipărește 2.
<code>y=sqrt(x);</code>	Atribuire variabilei y valoarea returnată de către funcția <code>sqrt(x)</code> . Rezultă $y=2$.
<code>z=(1+sqrt(x))/2;</code>	Valoarea returnată de funcția <code>sqrt</code> este folosită într-o expresie. Mai exact, valoarea expresiei $\frac{1+\sqrt{x}}{2}$ este atribuită variabilei z , rezultând $z=(1+\sqrt{4})/2=1.5$.
<code>t=floor(sqrt(x));</code>	Valoarea returnată de către funcția <code>sqrt(x)</code> , adică x^2 , este transmisă ca parametru funcției <code>floor</code> , iar rezultatul întors de către funcția <code>floor</code> este atribuit variabilei t . Așadar $t=floor(sqrt(4))=floor(16)=16$.

Aplicație

R.III.10. Valoarea unei expresii

Expresia `floor(-47 % 5 - sqrt(2)) / ceil(19.3) / 3` are valoarea ...
 (se consideră că $\sqrt{2} \approx 1.41$)

Rezolvare

Expresia dată este de forma $a / b / c$. Într-o astfel de expresie, împărțirile se evaluează de la stânga la dreapta, deci mai întâi se calculează a / b , apoi rezultatul este împărțit la c .

- Pentru expresia `floor(-47 % 5 - sqrt(2))` avem: $-47 \% 5 = -2$ (restul împărțirii lui -47 la 5), $\sqrt{2} \approx 1.41$, deci număratorul este `floor(-47 % 5 - sqrt(2)) = floor(-2 - 1.41) = floor(-3.41) = -4` (trunchiază numărul -3.41 la cel mai apropiat întreg mai mic decât el);
- `ceil(19.3) = 20` (rotunjește 19.3 la cel mai apropiat întreg mai mare decât el);
- prin urmare, expresia din enunț se reduce la $-4 / 20 / 3$, adică $-0.2 / 3$, respectiv -0.06667 .

Ați înțeles?

Probleme cu răspuns scurt

► 1. (Bacalaureat iulie 2008, varianta 18)

Care este valoarea tipărită de secvența de program de mai jos?

```
int a=25, b=7; float f;
f=(float)a/b;
f=(int)(f*100);
f=f/100;
cout << f;
printf("%0.2f", f);
```

a) 3
b) 3.57
c) 3.5714
d) 35.71

► 2. (Bacalaureat iulie 2009, varianta 74)

Care dintre următoarele expresii C++ are valoarea 1 dacă variabila x memorează un număr natural pătrat perfect?

- a) `sqrt(x) == floor(sqrt(x));` b) `sqrt(x) != floor(sqrt(x));`
 c) `floor(sqrt(x)) != ceil(sqrt(x));` d) `x - floor(x) == ceil(x);`

► 3. (Bacalaureat iulie 2009, varianta 20)

Care este valoarea tipărită de secvența de program C++ alăturată, știind că x și y sunt variabile întregi, iar z este o variabilă reală?

```
x=30;
y=5;
z=(x+y)/6.0;
z=floor(z*100);
z=z/100;
cout << z;
printf("%0.2f", z);
```

a) 5
b) 5.833333
c) 5.83
d) 583

► 4. (Bacalaureat iulie 2009, varianta 8)

Care dintre următoarele expresii C++ are ca valoare cel mai mare dintre numerele naturale nenule, memorate în variabilele întregi a și b ?

- a) `(a+b+abs(a-b))/2;` b) `a+b+abs(a-b)/2;`
 c) `(a+b-abs(a-b))/2;` d) `(a+b-abs(a+b))/2;`

► 5. Instrucțiunea `"cout << floor(27 % 4) + ceil(27.3) / 4;"` afișează valoarea

► 6. Fiind dată variabila x cu valoarea -0.2 , precizați care dintre expresiile logice de mai jos au valoarea 1:

- a) `floor(x+0.5) == ceil(x)` b) `floor(x-0.5) == ceil(x)`
 c) `floor(x)+1 == ceil(x)` d) `floor(x)-1 == ceil(x)`
 e) `floor(x) == ceil(x)` f) `floor(x+1) == ceil(x-1)`

► 7. (Bacalaureat iulie 2001, varianta 2)

Pentru a atribui variabilei x rezultatul expresiei $\sqrt{a^2 - b^2}$, unde a și b desemnează două variabile reale, se utilizează instrucțiunea de atribuire:

- a) $x = \sqrt{\sqrt{a}} - \sqrt{b}$
 c) $x = \sqrt{a \cdot a} - b \cdot b$

- b) $x = \sqrt{a \cdot a - b \cdot b}$
 d) $x = \sqrt{a + b} \cdot (a - b)$

► 8. Știind că $n=3$ și $m=5$, precizați care instrucțiuni sunt echivalente:

- I) $p = \text{floor}(m/n + n/m)$; II) $p = \text{floor}((m \cdot m + n \cdot n) / (m \cdot n))$;
 III) $p = \text{floor}(m/n) + \text{floor}(n/m)$;
 a) I și II b) I și III c) II și III d) toate

► 9. Fie variabila x de tip real, cu valoarea 16.3, și instrucțiunile I1, I2 și I3 de mai jos. Care dintre aceste instrucțiuni afișează 40.3?

- I1. `cout << "\n" << sqrt(floor(x)) << x-floor(x);`
 I2. `cout << "\n" << 40+abs(ceil(x)-x);`
 I3. `cout << "\n" << 3*floor(x)+0.3-floor(x/2);`

- a) I1 și I2 b) I2 și I3 c) numai I1 d) toate

► 10. Știind că de la tastatură citim în variabila x un număr real *pozitiv*, ce reprezintă valoarea variabilei t pe care o afișează secvența următoarea de instrucțiuni?

```
cin >> x;
y=x*100;
z=floor(y);
t=z/100;
cout << t;
```

- a) Numărul obținut din x prin eliminarea ultimei zecimale;
 b) Numărul x trunchiat la primele două zecimale;
 c) Partea întreagă a lui x ;
 d) Partea zecimală a lui x ;



III.4.4. Tipuri ne-numerice

Tipul caracter (char)

O variabilă de tip caracter ia ca valoare un caracter cuprins între apostrofuri și se declară cu cuvântul `char`.

Exemplu: `char c;`

Pentru a citi de la tastatură un caracter, putem folosi oricare din funcțiile `getchar()`, `getch()`, `getc()` și `getche()`. Toate aceste funcții citesc un caracter și returnează caracterul citit. Pentru a memora într-o variabilă caracterul citit, vom atribui variabilei valoarea returnată de către funcția de citire.

Exemplu:

Dacă vrem să citim un caracter într-o variabilă c de tipul `char`, vom scrie în program o instrucțiune de genul

`c=getchar();`

Deosebiriile dintre funcțiile de citire sunt următoarele:

- Funcția `getchar()` preia caracterul numai după apăsarea tastei `ENTER` și îl afișează pe ecran (vom spune că citirea se face "cu ecou");
- Funcția `getche()` preia caracterul fără să mai aștepte apăsarea tastei `ENTER`, și îl afișează (adică citirea se face "fără ecou");
- Funcția `getc()` preia caracterul fără să mai aștepte "confirmarea" cu `ENTER`, dar nu îl afișează.

O variabilă de tipul caracter poate fi citită însă și cu ajutorul binecunoscutei instrucțiuni `cin`.

Exemplu:

Pentru aceeași variabilă c de tipul `char` declarată mai sus, putem realiza următoarea citire de la tastatură:

`cin >> c;`

Afișarea unui caracter se poate face de asemenea în două moduri:

- Cu una din funcțiile `putch`, `putc`, `putchar`, care primesc drept parametru caracterul ce trebuie afișat (o constantă-caracter sau o variabilă de tipul `char`);
- Cu ajutorul instrucțiunii `cout`.

Exemplu:

```
putchar('m');
putchar(c); // unde c este o variabilă de tipul char
```

Și în ceea ce privește tipul caracter, limbajul C++ are o particularitate deosebită, și anume faptul că realizează conversii implicite, de la caracter la întreg și invers. Astfel, o variabilă de tip întreg poate lua ca valoare un caracter, iar o variabilă de tip caracter poate primi ca valoare un număr întreg. Aceste conversii implicite se realizează prin intermediul așa-numitelor coduri ASCII ale caracterelor.

Trebuie însă reținut că o astfel de conversie poate genera o pierdere de informație în anumite cazuri, lucru care se va exemplifica în continuare.

Reamintim:

Limbajul C++ folosește codificarea *ASCII* a caracterelor. În cadrul setului *ASCII*, fiecare caracter ocupă o poziție, adică are un număr de ordine, numit *codul ASCII al caracterului*. Codurile *ASCII* sunt numerele naturale de la 0 la 255. Literele mari 'A', 'B', ..., 'Z' au codurile cuprinse între 65 și 90, codurile literelor mici 'a', 'b', ..., 'z' sunt de la 97 la 122, caracterele-cifre '0', '1', ..., '9' au codurile între 48 și 57 inclusiv, iar caracterele speciale sunt intercalate în tot intervalul [0, 255].

Odată reamintită noțiunea de cod *ASCII*, putem enunța aspectele care stau la baza conversiilor implicite *întreg* → *caracter* și *caracter* → *întreg*:

- Unei variabile de tip caracter i se poate atribui un număr întreg; în acest caz, valoarea variabilei va fi caracterul ce are drept cod ASCII întregul atribuit (întregul se convertește implicit la caracter).

Exemplu:

Fie secvența de program:

```
char c1;
c1=68;
putchar(c1);
```

Atribuirea "c1=68;" se execută astfel: numărul întreg 68 este convertit automat la caracterul cu codul ASCII 68, în speță caracterul 'D', iar acest caracter ca fi atribuit variabilei c1 de tipul char. Prin urmare c1='D', deci funcția putchar afișează caracterul 'D'.

- Analog, unei variabile de tip întreg i se poate atribui un caracter; în această situație, valoarea variabilei va fi codul *ASCII* al caracterului atribuit (caracterul se convertește la întreg).

Exemplu:

Fie secvența de program:

```
int m,n;
m='D';
cout << "\n" << m;
putchar(m);
n='E'-'B';
cout << "\n" << n;
```

Cum funcționează atribuirea "m='D';"? Caracterul 'D' este convertit automat la codul său ASCII care este 68, iar acest întreg 68 este atribuit variabilei m de tipul int. Prin urmare, instrucțiunea cout tipărește numărul 68.

Atenție însă, instrucțiunea "putchar(m);" va afișa caracterul 'D' ! De ce ? Parametrul funcției putchar trebuie să fie de tip caracter, deci " în interiorul " funcției are loc o conversie inversă, de la întreg la caracter !

Deși aparent ciudată, atribuirea "n='E'-'B';" este foarte corectă. "Scăderea între două caractere" înseamnă de fapt diferența codurilor lor ASCII. În consecință, n=69-66 (codul lui 'E' minus codul lui 'B'), deci n=3.

Ați înțeles ? Probleme cu răspuns scurt

► 1. (Bacalaureat iulie 2008, varianta 24)

Care dintre expresiile următoare, scrise în limbajul C++, are valoarea 1?

- a) 'a'==A b) '1'+'2'=='3' c) 'a'<'b' d) '1'>'2'

- 2. În programul următor, care dintre secvențele de instrucțiuni notate cu (1), (2) și (3) nu vor produce erori la execuție ?

```
#include <stdio.h>
void main ()
{
    int x,y; char m;
    m='A'; putchar(m+1); // (1)
    x=getchar(); m='A'; x-=m; printf("%c %d ",x,x); // (2)
    x=300; m=x; y=m; printf("\n%c %d\n",y,y); // (3)
}
```

- a) Numai secvențele (1) și (3) b) Toate trei c) Nici una
d) Numai secvențele (1) și (2)

► 3. Funcțiile getchar(), getch() și getche() citesc de la tastatură un caracter. Ce deosebiri există între cele trei funcții?

- a) Funcțiile getchar și getche realizează citirea cu ecou, iar getch citește caracterul fără ecou.
b) Funcția getchar citește caracterul cu ecou, iar funcțiile getche și getch realizează citirea fără ecou.
c) Funcțiile getchar și getch preiau caracterul numai după apăsarea tastei ENTER.
d) Funcțiile getchar și getche preiau caracterul de îndată ce a fost tastat, fără să mai aștepte "confirmarea" cu ENTER.
e) Toate cele trei funcții au prototipul în header-ul conio.h.

► 4. Precizați valoarea lui n, rezultată în urma execuției programului:

```
#include <iostream.h>
void main()
{
    char c; int n=97;
    n=c-n;
}
```

- a) Programul este greșit b) 97 c) 99 d) 79 e) NULL

► 5. Precizați valoarea pe care o va avea variabila c în urma execuției programului de mai jos:

```
#include <iostream.h>
void main()
{
    char c='d';
    int n=99;
    c=n+1=c-1;
}
```

- a) 'd' b) 'c' c) 'b' d) atribuirea este greșită



III.5. Instrucțiuni pentru lucrul cu ecranul

Versiunile de DOS ale limbajului C++ conțin câteva funcții predefinite pentru lucrul cu ecranul monitorului: ștergerea ecranului, stabilirea culorilor, generarea sunetelor etc. Acestea nu se află în nucleul limbajului, ci sunt grupate într-un header, și anume `conio.h`. Dacă dorim să apelăm funcții din această bibliotecă, trebuie să includem în program header-ul `conio.h`, scriind linia de program `#include <conio.h>` la începutul programului.

Ștergerea ecranului se face apelând funcția `clrscr()` ("clear screen"), care nu are parametri.

Ecranul este împărțit în 25 de linii și 80 de coloane. Fiecare punct de pe ecran se caracterizează prin două coordonate: linia și coloana pe care se află. Pentru a poziționa cursorul pe coloana `<col>` și linia `<lin>`, folosim funcția `gotoxy`.

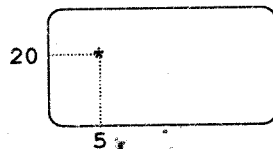
Sintaxă:

`gotoxy (<col>, <lin>);`

Exemplu:

```
#include <iostream.h>
#include <conio.h>
```

```
void main()
{
    clrscr();
    gotoxy(5,20);
    cout << "\n test";
}
```



În C++ sunt disponibile 16 culori. Fiecare culoare este identificată în două moduri:

- printr-un cod numeric, un număr întreg din intervalul [0,15];
- printr-o constantă simbolică.

Pentru a scrie cu o anumită culoare pe ecran, apelăm funcția `textcolor`.

Sintaxă:

`textcolor(<culoare>);`

- ♦ `<culoare>` → un cod de culoare sau o constantă simbolică atașată unei culori.

Culoarea fondului se stabilește apelând funcția `textbackground`.

Sintaxă:

`textbackground(<culoare>);`

Pentru a șterge și a colora ecranul în întregime cu culoarea `c`, vom face un apel `textbackground(c)` urmat de un apel `clrscr`.

Efectul de "clipire" se obține adăugând valoarea 128 sau constanta simbolică `BLINK` la codul culorii.

Exemplu:

- `textbackground (CYAN)` → stabilește culoarea fondului *turcoaz*
- `textcolor (4)` → stabilește culoarea de scriere *roșu*
- `textcolor (1+BLINK)` → stabilește culoarea de scriere *albastru*, cu clipire.



Expresii condiționale

Expresiile condiționale au rolul de a introduce "ramificații" în program. Se testează o anumită condiție, și, în funcție de valoarea de adevăr a expresiei logice care descrie condiția (zero sau diferit de zero), se alege o "acțiune" din două posibile.

Sintaxă:

`<expr_1> ? <expr_2> : <expr_3>;`

Execuția unei expresii condiționale se produce în felul următor:

- ♦ se evaluează mai întâi expresia logică `<expr_1>`:
- dacă valoarea acesteia este diferită de 0, atunci valoarea și tipul expresiei condiționale sunt date de expresia `<expr_2>`;
- în caz contrar, valoarea și tipul expresiei condiționale sunt date de expresia `<expr_3>`.

Operatorii condiționali sunt "?" și ":". Observăm că o expresie condițională are ca efect "ramificarea" programului: dacă `<expr_1>` are valoarea diferită de zero se execută `<expr_2>`, în caz contrar se efectuează `<expr_3>`.

Exemplu:

`y>0 ? x=a : x=b;`

- `<expr_1>` este expresia logică "`y>0`", `<expr_2>` este atribuirea `x=a`, iar `<expr_3>` este atribuirea `x=b`.

Pentru exemplificare, să urmărim efectul execuției expresiei, pentru `a=15`, `b=20`, `y=3`: expresia logică "`y>0`" are valoarea 1 ("`3>0`?", *adevărat*), deci se execută atribuirea `x=a`, rezultând `x=15`. Expresia condițională primește valoarea expresiei executate, adică 15.

Se observă că expresia condițională de mai sus descrie enunțul: "dacă numărul y este pozitiv, atunci x ia valoarea lui a , în caz contrar x ia valoarea lui b ".

În cazul în care expresiile $\langle \text{expr}_2 \rangle$ și $\langle \text{expr}_3 \rangle$ aferente celor două "ramuri" constau în atribuire de valori pentru aceeași variabilă, se poate folosi o formă particulară de expresie condițională. Sintaxa este:

Sintaxa:

$\langle \text{var} \rangle = \langle \text{expr}_1 \rangle ? \langle \text{expr}_2 \rangle : \langle \text{expr}_3 \rangle ;$

Funcționarea în acest caz este următoarea:

- ♦ se evaluează expresia logică $\langle \text{expr}_1 \rangle$:
 - dacă aceasta are o valoare diferită de zero, atunci variabilei $\langle \text{var} \rangle$ i se atribuie expresia $\langle \text{expr}_2 \rangle$;
 - în caz contrar, variabilei $\langle \text{var} \rangle$ i se atribuie expresia $\langle \text{expr}_3 \rangle$.

Exemplu:

Folosind această a doua formă, expresia condițională " $y > 0 ? x = a : x = b$;" din exemplul anterior, se poate scrie și astfel:

$x = (y > 0) ? a : b$

- variabila $\langle \text{var} \rangle$ este x , expresia logică $\langle \text{exp}_1 \rangle$ este " $y > 0$ ", $\langle \text{expr}_2 \rangle$ este " a ", iar $\langle \text{expr}_3 \rangle$ este " b ".

Expresia se "traduce" astfel: dacă $y > 0$, atunci x ia valoarea lui a , în caz contrar x ia valoarea lui b , adică exact enunțul de la prima formă.

Menționăm că parantezele puteau să lipsească, deoarece operatorii "?" și ":" ai unei expresii condiționale au prioritate mai mică decât toți operatorii aritmetici, logici și relaționali. Așadar, se putea scrie: " $x = y > 0 ? a : b$;".

Iată acum un exemplu în care una dintre "ramurile" unei expresii condiționale este tot o expresie condițională.

Exemplu: $a > 0 ? (b > 0 ? (x = 8) : (x = 5)) : (x = 6) ;$

Interpretarea expresiei condiționale este următoarea:

- dacă $a > 0$, se execută expresia de după operatorul "?", care este tot o expresie condițională: " $b > 0 ? (x = 8) : (x = 5)$ "; aceasta la rândul ei înseamnă: "dacă $b > 0$ " atunci x ia valoarea 8, în caz contrar x ia valoarea 5;
- în caz contrar, respectiv pentru $a \leq 0$, se execută expresia de după operatorul ":", adică atribuirea " $x = 6$ ".

Funcția (operatorul) `sizeof`

Funcția `sizeof` are două forme:

- `sizeof(<tip_d>)` returnează numărul de octeți necesar memorării valorilor de tipul $\langle \text{tip}_d \rangle$ (dimensiunea în octeți a tipului de date $\langle \text{tip}_d \rangle$);

- `sizeof(<exp>)` returnează numărul de octeți necesari memorării valorii expresiei $\langle \text{exp} \rangle$.

Exemplu:

```
int x=5,y;
y=(sizeof(x-1)==sizeof(int)) ? sizeof('x') : sizeof(3);
cout << y;
```

Observați că în a doua linie a programului avem o expresie condițională a cărei "protagonistă" este variabila y .

Mai întâi se evaluează expresia logică "`sizeof(x-1)==sizeof(int)`":

- dacă această expresie are valoarea 1 ("adevărat"), atunci are loc atribuirea $y = \text{sizeof}('x')$;
- în caz contrar, se execută atribuirea $y = \text{sizeof}(3)$.

Având în vedere modul de lucru al funcției `sizeof`, rezultă că valoarea de adevăr a condiției este "adevărat" deci va fi evaluată expresia `sizeof('x')`, iar y va primi această valoare. Cum pentru memorarea unui caracter este necesar un octet, indiferent de tipul de calculator și sistemul de operare folosit, rezultă că y va primi valoarea 1, valoare care se afișează cu instrucțiunea `cout`.

Facem precizarea că nu la fel ar fi stat lucrurile, dacă în loc de `sizeof('x')` am fi considerat expresia `sizeof(x)`. Spre deosebire de memorarea caracterelor, care se face întotdeauna pe un octet, variabilele de tip `int` pot fi memorate diferit, în funcție de context (calculator, sistem de operare, mediu de programare etc.). Astfel, de exemplu, compilatoarele Borland C++ până la versiunea 4 inclusiv folosesc memorarea variabilelor de tip `int` pe doi octeți în vreme ce mediile vizuale, începând de la 5 inclusiv, folosesc reprezentarea tipului `int` pe 4 octeți. Acesta este motivul pentru care programatorii folosesc funcția `sizeof` ori de câte ori fac apel la reprezentarea unei variabile, în loc de o formulare explicită de forma 2 sau 4 etc. Se obțin pe această cale programe portabile, a căror funcționare nu mai este dependentă de mediul în care se execută.

Operatorul "virgulă"

O expresie compusă cu operatorul "virgulă" este de fapt o înșiruire de expresii separate între ele prin caracterul "virgulă", pe care le numim expresii elementare.

Sintaxa:

$\langle \text{exp}_1 \rangle, \langle \text{exp}_2 \rangle, \langle \text{exp}_3 \rangle, \dots$

Execuția acestei expresii compuse este foarte simplă: se execută succesiv expresiile elementare, de la stânga la dreapta.

Exemplu:

```
int a=2,b,c;
b=2*a+1, b++, c=b*b;
cout << "\n" << a << " " << b << " " << c;
```

În a doua linie a programului, avem o expresie compusă cu operatorul "virgulă", alcătuită din expresiile elementare " $b = 2 * a + 1$ ", " $b++$ " și " $c = b * b$ ". Aceste expresii elementare sunt evaluate succesiv, de la stânga la dreapta. Așadar:

- $b=2*a+1 \Rightarrow b=2*2+1=5;$
- $b++ \Rightarrow b=5+1=6;$
- $c=b*b \Rightarrow c=6*6=36.$

Prin urmare, instrucțiunea cout tipărește numerele 2, 6 și 36, cu câte un spațiu între ele.

O expresie compusă cu operatorul "virgulă" poate fi atribuită unei variabile. La finele execuției sale, expresia compusă "întoarce" valoarea ultimei expresii elementare componente (cea care se execută ultima, adică cea mai din dreapta). Dacă expresia compusă este atribuită unei variabile, atunci variabilei i se atribuie de fapt valoarea ultimei expresii elementare.

Într-o astfel de atribuire, pentru a evita orice ambiguități, este recomandabil ca expresia compusă atribuită să o includem între paranteze.

Sintaxa:

<var> = (<exp_1>, <exp_2>, <exp_3>....)

- unde <var> este identificatorul variabilei căreia i se atribuie expresia compusă cu operatorul "virgulă".

Exemplu:

```
int a=10, b=6, c=4, d;
d=(c=a-6, a=b*c, b+=a, a/2);
printf("\n%d %d %d %d", a, b, c, d);
```

Expresia compusă în sine este "c=a-6, a=b*c, b+=a, a/2", și se atribuie variabilei d. Expresiile elementare se execută succesiv, de la stânga la dreapta:

- $c=a-6 \Rightarrow c=10-6, c=4;$
- $a=b*c$ (restul împărțirii întregi a lui b la c) $\Rightarrow a=6*4, a=24;$
- $b+=a \Rightarrow b=b+a$ (operatorul "+=") $\Rightarrow b=6+24, b=30;$
- $a/2$: aceasta este o expresie aritmetică cu valoarea 1 (câtul împărțirii întregi a lui 2 la 2). Ea se atribuie variabilei d, pentru că este ultima dintre expresiile elementare ce alcătuiesc expresia compusă. Rezultă d=1.

Valorile finale ale variabilelor, pe care le afișează instrucțiunea printf, vor fi a=2, b=30, c=4, d=1.

Dacă ultima expresie elementară ar fi fost tot o atribuire, se mai putea atribui expresia compusă variabilei d? Răspunsul este afirmativ. Iată un exemplu:

```
d=(c=a-6, a=b*c, b+=a, b=a/2);
```

Primele trei expresii elementare sunt identice cu cele de mai sus, în urma execuției lor rezultând în ordine c=4, a=2 și b=8. Ultima expresie elementară este atribuirea b=a/2. Se atribuie astfel variabilei b valoarea 1. Dar valoarea atribuită 1 este valoarea ultimei expresii elementare b=a/2, deci ea este "întoarsă" de către expresia condițională. În consecință, variabila d primește valoarea 1, ca și în cazul anterior.



Probleme propuse

►1. Scrieți un program care citește de la tastatură două valori reprezentând înălțimea și greutatea unei persoane, apoi tipărește pe ecran:

Eu am ... kg greutate
și ... m înălțime

unde în loc de "..." se vor scrie valorile citite anterior.

►2. Să se scrie un program care citește de la tastatură cele două catete c1 și c2 ale unui triunghi dreptunghic, apoi calculează și afișează celelalte mărimi ale triunghiului: ipotenuza, sinusurile și cosinusurile unghiurilor ascuțite, înălțimea corespunzătoare unghiului drept și aria triunghiului.

►3. O persoană are la bancă la începutul anului o sumă de bani S. Cunoscând faptul că pe primele șase luni dobânda a fost de p %, iar pe următoarele șase luni de q % (aplicată sumei existente după primele șase luni), să se afle ce sumă avea persoana la sfârșitul anului.

►4. Un bloc de locuințe are formă paralelipipedică, cu laturile bazei de x respectiv y dam, și înălțimea de y metri. Pentru construcția lui se folosesc cărămizi de forma cubică, având latura de a metri. Cărămizile sunt livrate firmei constructoare în cutii de câte 100 de bucăți, fiecare cutie costând L lei. Să se afle numărul cărămizilor folosite pentru ridicarea blocului, precum și prețul total al acestora.

►5. Un cetățean care dorește să aibă o livadă, și-a cumpărat un număr de p pomi pe care dorește să-i planteze pe terenul de care dispune. Știind că x % din totalul pomilor sunt meri, y % din pomii rămași după plantarea merilor sunt pruni, iar restul pomilor sunt peri, să se afle câți pomi de fiecare tip a plantat cetățeanul nostru.

►6. Într-o clasă sunt e elevi. Știind că numărul băieților din clasă este cu b mai mare decât numărul fetelor, afișați câte fete și câți băieți sunt în clasă.

►7. Realizați un program care afișează, sub o formă grafică aspectuoasă, un tabel cu valorile funcțiilor trigonometrice SIN, COS, TG și CTG, pentru unghiurile uzuale din primul cadran al cercului trigonometric: 0°, 30°, 45°, 90°.

►8. Se consideră trei variabile reale a, b și c ale căror valori se citesc de la tastatură. Se cere să se interschimbe circular de la stânga la dreapta valorile lor, apoi să se afișeze valorile astfel interschimbate. Exemplu: dacă inițial a=3, b=16, c=4.2, atunci după interschimbare vom avea a=4.2, b=3, c=16.

Indicații. Valorile variabilelor se vor permuta astfel: a→b ("a trece în b"), b→c, c→a. Pentru a nu pierde însă valorile inițiale, trebuie să folosim variabile intermediare.

►9. Realizați un program care citește de la tastatură un număr natural n , apoi calculează suma primelor n numere naturale $S=1+2+\dots+n$. Exemplu: pentru $n=7$ rezultă $S=1+2+3+4+5+6+7=28$.

Indicații. Pentru a calcula suma cerută, adunăm câte doi termeni egal depărtați de extreme: 1 cu n , 2 cu $n-1$, ș.a.m.d. Pe exemplul dat, avem: $1+7=8$, $2+6=8$, $3+5=8$. Observăm că toate sumele dau același rezultat egal cu cel al primeia adică $1+n$, și în total avem $[n/2]$ sume (prin "[]" am simbolizat partea întreagă).

►10. Pentru două numere naturale n și k citite de la tastatură, scrieți un program care calculează expresia $k+(k+1)+(k+2)+\dots+(k+n)$.

Indicații. Printr-un artificiu de calcul, scriem $k+(k+1)+(k+2)+\dots+(k+n)=(1+2+\dots+n)-(1+2+\dots+k)$. Mai departe folosiți indicațiile de la problema anterioară.

►11. Realizați un program care citește un număr natural x cu exact patru cifre, apoi afișează media aritmetică a cifrelor sale.

Indicații. Inițializăm cu 0 o variabilă S în care vom memora suma celor patru cifre ale lui x . Extragem ultima cifră a numărului x , ca fiind restul împărțirii lui x la 10 și adăugăm cifra la S -ul anterior. Apoi câtul împărțirii îl împărțim din nou la 10 și noul rest va reprezenta penultima cifră, adăugând-o și pe aceasta la S . Procedăm analog pentru cifrele rămase, iar în final media aritmetică a cifrelor va fi egală cu suma cifrelor împărțită la 4.

►12. Se citește de la tastatură un număr natural de patru cifre. Să se afișeze numerele obținute prin aplicarea următoarelor operații:

- schimbarea primei cifre cu ultima;
- schimbarea cifrelor din mijloc între ele.

Exemplu: pentru numărul natural 5629, se vor afișa mai întâi 9625, apoi 5269.

Indicații. Extragem cele patru cifre ale numărului printr-un algoritm similar cu cel descris în indicațiile de la problema anterioară, cifre pe care le memorăm în patru variabile. Apoi aplicăm algoritmul cunoscut de interschimbare cu "metoda scaunelor" între prima și ultima variabilă, respectiv a doua și a treia. În final afișăm succesiv cele patru variabile pe un singur rând fără spații între ele, simulând astfel afișarea numărului rezultat.



Capitolul IV: Instrucțiunile limbajului

Precizare: În secțiunile de "Probleme rezolvate" din acest capitol, vom relua toate problemele prezentate în cadrul capitolului "Principiile programării structurate". Întrucât în capitolul amintit algoritmi au fost explicați în detaliu, considerăm că este suficient să reamintim pseudocodul și să prezentăm textul programului, punctând doar câteva comentarii suplimentare acolo unde este cazul.



IV.1. Instrucțiuni de selecție

IV.1.1. Instrucțiunea de selecție simplă

Instrucțiunea de selecție simplă **if-se** este transcrierea în limbajul Pascal a structurii de selecție simplă învățată în pseudocod.

```
dacă (<expresie>) atunci
  <secvența_1>
altfel
  <secvența_2>
```

```
if (<expresie>)
  <secvența_1>
else
  <secvența_2>
```

- ◆ Expresia logică **<expresie>** trebuie scrisă întotdeauna între paranteze rotunde.
- ◆ Se testează condiția dată de expresia logică **<expresie>**:
 - ✓ dacă aceasta este adevărată, se execută secvența de instrucțiuni **<secvența_1>** (corpul ramurii "if");
 - ✓ în caz contrar, se execută **<secvența_2>** (corpul ramurii "else").

Exemplu: Maximul a două numere întregi a și b citite de la tastatură.

Se citesc de la tastatură numerele a și b . Apoi se compară numerele, memorându-se cel mai mare dintre ele în variabila max : dacă $a > b$ atunci maximul este a , în caz contrar maximul este b . În final se afișează maximul max .

```
citește a,b;
dacă (a>b) atunci
  max ← a;
altfel
  max ← b;
scrie max;
```

```
#include <iostream.h>
void main()
{
  int a,b,m;
  cout << "\n a,b=";
  cin >> a >> b;
  if (a>b)
    max=a;
  else
    max=b;
  cout << "max= << max;
end.
```


Observații

Ramura **else** împreună cu secvența aferentă poate să lipsească. În această situație nu se execută nimic pe "cazul contrar" (dacă condiția dată de <expresie> este falsă).

Dacă secvența de instrucțiuni <secvența_1> sau <secvența_2> conține mai mult de o instrucțiune, ea trebuie cuprinsă între acoladă deschisă și acoladă închisă.

Într-o instrucțiune **if-else**, corpul oricărei ramuri poate fi la rândul său o altă instrucțiune **if-else**, caz în care se formează o așa-numită "structură **if-else** imbricată". Implicit, fiecare **else** se asociază ultimului **if** anterior care nu a fost asociat, așa cum arată și schema alăturată. Dacă dorim să schimbăm această asociere, vom folosi acoladele.

```
if <c1> then
  if <c2> then
    <s1>
  else
    <s2>
else
  <s3>
```

Exemplu:

```
{ Varianta A }
x=1;
if (!(a==b) ||
    !(a==c))      {(1)}
if (b>c)          {(2)}
    cout << -x    {(3)}
else              {(4)}
    if (b<c)      {(5)}
        cout << x; {(6)}
```

```
{ Varianta B }
x=1;
if (!(a==b) ||
    !(a==c))      {(1)}
{
    if (b>c)       {(2)}
        cout << -x {(3)}
}
else              {(4)}
    if (b<c)       {(5)}
        cout << x; {(6)}
```

În varianta A:

- instrucțiunea **else** din linia (4) se asociază **if**-ului din linia (2) (ultimul înainte de ea). Corpul acestui **else** este alcătuit din liniile (5) și (6), adică o nouă instrucțiune **if**. Toate liniile (2),..., (6) reprezintă corpul **if**-ului din linia (1).

În varianta B:

- prezența acoladelor deschise și închise schimbă lucrurile: instrucțiunea **else** din linia (4) se asociază **if**-ului din linia (1). Corpul acestui **if** este un alt **if**, și anume liniile (2), (3), și este cuprins între acolade, pentru că altfel s-ar face asocierea implicită din varianta cealaltă. În fine, corpul lui **else** din linia (4) este un alt **if**, în speță liniile (5) și (6).

Aplicație R.IV.1. Aria unui triunghi

Scrieți un program care citește de la tastatură două numere reale și strict pozitive **b** și **h**, reprezentând baza, respectiv înălțimea unui triunghi, apoi calculează și afișează aria triunghiului. În cazul în care valorile introduse nu pot reprezenta latura și înălțimea unui triunghi, se va afișa mesaj de eroare. Reamintim formula ariei triunghiului: $A = (b \cdot h) / 2$.

Rezolvare

Pseudocodul acestui algoritim a fost descris pe larg în capitolul I, lecția "Primul exemplu de pseudocod". Se citesc de la tastatură valorile bazei **b** și înălțimii **h** ale triunghiului. Dacă acestea sunt ambele pozitive {"(b>0) și (h>0)"}, atunci calculează și afișează aria triunghiului. În caz contrar tipărește un mesaj de eroare. Am prezentat două variante:

a) valoarea ariei se memorează prin atribuire într-o variabilă **A**.

```
#include <iostream.h>

void main ()
{
    int b,h; float A;
    cout << "Dati baza si inaltimea ";
    cin >> b >> h;
    if (b>0 && h>0)
    {
        A=(b*h)/2.;
        cout << "\n Aria= " << A;
    }
    else
        cout << "\n Date incorecte ";
}
```

```
citește b,h;
dacă (b>0 și h>0)
    început
        A←(b*h)/2;
        scrie A;
    sfârșit
    altfel
        scrie 'Date incorecte ';
```

Întrucât dorim să se obțină valoarea exactă a ariei triunghiului, ca număr real cu parte zecimală, trebuie să declarăm variabila **A** ca fiind de tipul **float** (chiar dacă **b** și **h** sunt numere întregi, declarate de tipul **int**), dar acest lucru nu este suficient: instrucțiunea de atribuire a valorii ariei trebuie scrisă sub forma " $A = (b \cdot h) / 2.$ ", și nu " $A = (b \cdot h) / 2$ ". Punctul zecimal atașat constantei 2 determină împărțirea lui $(b \cdot h)$ la numărul real 2.0, rezultând astfel tot un număr real cu parte întreagă și parte zecimală. Dacă am fi scris " $A = (b \cdot h) / 2$ ", atunci compilatorul ar fi făcut împărțirea întreagă a număratorului $(b \cdot h)$ la 2, reținând doar partea întreagă a rezultatului!

b) valoarea ariei este afișată direct, ca expresie.

```
#include <iostream.h>

void main ()
{
    int b,h; float A;
    cout << "\n Dati baza si inaltimea ";
    cin >> b >> h;
    if (b>0 && h>0) cout << "\n Aria= " << (b*h)/2.;
    else cout << "\n Date incorecte ";
}
```

```
citește b,h;
dacă (b>0 și h>0)
    scrie (b*h)/2 ;
    altfel
        scrie 'Date incorecte ';
```

Încercați singuri!

►1. Scrieți programe pentru algoritmi propuși în tema T.1. din cadrul lecției "II.3.1 Structura de selecție simplă" (capitolul II).

Aplicație R.IV.2. Funcție matematică

Să se calculeze valoarea funcției matematice $f(x)$, pentru o valoare a lui x introdusă de la tastatură:

$$f: \mathbb{R} \rightarrow \mathbb{R}, f(x) = \begin{cases} x^2 + 1, & \text{pentru } x \leq -3 \\ x - 2, & \text{pentru } x \in (-3, 3) \\ x^2 - 4x + 5, & \text{pentru } x \geq 3 \end{cases}$$

Rezolvare

Pseudocodul acestui algoritm a fost explicat pe larg în cadrul aplicației R.II.3. din cadrul lecției "II.3.1 Structura de selecție simplă" (capitolul II).

Pseudocodul

```

citește x;
dacă (x<=-3) atunci
    E←x*x+1
altfel
    dacă (x<3) atunci
        E←x-2
    altfel
        E←x*x-4*x+5;
scrie E;
    
```

Programul

```

#include <iostream.h>
#include <conio.h>

void main ()
{
    float E,x;
    clrscr();
    // citește o valoare a argumentului x
    cout << "x=";
    cin >> x;
    // memorează în variabila E valoarea funcției aleasă conform valorii lui x
    if (x<=-3)
        E=x*x+1;
    else
        if (x<3)
            E=x-2;
        else
            E=x*x-4*x+5;
    // afișează valoarea funcției indiferent de "ramura" aleasă
    cout << endl << "E=" << E;
    getch();
}
    
```

Încercați singuri!

► 1. Scrieți programe pentru algoritmi propuși în tema T.3. din cadrul lecției "II.3.1 Structura de selecție simplă" (capitolul II).

Aplicație R.IV.3. Problemă cu răspuns scurt

(Bacalaureat iulie 2009, varianta 95)

În secvența de instrucțiuni de mai jos, variabilele i , j , k , x și y sunt de tip întreg. Pentru care dintre următoarele seturi de valori ale variabilelor i , j și k , variabilele x și y vor primi valori diferite în urma executării acestei secvențe?

```

if (k>0)
    if (i!=j)
        x=0;
    else
        x=1;
else
    x=2;
if (i!=j)
    if (k>0)
        y=0;
    else
        y=2;
else
    y=1;
    
```

- a) x și y primesc aceeași valoare, indiferent de valorile lui i , j și k
- b) $k=0, i=5, j=6$
- c) $k=10, i=5, j=5$
- d) $k=0, i=5, j=5$

Rezolvare

Secvența conține două instrucțiuni "if-else" imbricate. După cum am arătat în partea de teorie, într-o instrucțiune "if-else" imbricată, fiecare "else" se asociază primului "if" de dinaintea lui care nu a fost încă asociat. Din modul în care am așezat în pagină "ramurile" instrucțiunilor de selecție, vă puteți da seama care sunt asocierile. Astfel, pe prima structură "if-else" imbricată avem:

– ramura "else $x=1$;" se asociază primului "if" de dinaintea lui, adică lui "if ($i!=j$)";

– ramura "else $x=2$;" se asociază lui "if ($k>0$)".

În mod similar au loc asocierile și în cazul celei de-a doua structuri "if-else" imbricate, cea care pleacă de la "if ($i!=j$)".

Pentru început vom analiza variantele b) și d), care au ambele $k=0$. Întrucât condiția " $k>0$ " din primul "if" nu este îndeplinită, se va intra pe ramura "else" a acestuia, care conține atribuirea $x=2$. Mai departe, evoluția algoritmului pe cele două variante este diferită, și anume:

– În varianta b), cu $i=5$ și $j=6$, condiția din linia "if ($i!=j$)" este adevărată (5≠6), motiv pentru care intră pe ramura "if" a acestei structuri imbricate, unde găsește un alt "if-else". Aici condiția " $k>0$ " este falsă (noi am plecat de la $k=0$), deci va executa

ramura "else" a structurii interioare, unde avem atribuirea $y=2$. În concluzie, în varianta b) vom avea $x=2$ și $y=2$, adică valori egale.

– În schimb, în varianta d), cu $i=5$ și $j=5$, condiția din linia "if ($i!=j$)" este falsă, deci va executa ramura "else" a acestui "if-else", unde nu mai avem un alt "if-else", ci doar atribuirea $y=1$. Iată așadar că în varianta d) am obținut $x=2$ și $y=1$, adică valori diferite pentru cele două variabile de ieșire, ceea ce corespunde cerinței problemei.

Acum este foarte clar că varianta a) nu este corectă. Cât privește varianta c), vă lăsăm ca exercițiu să dovediți faptul că, pentru $k=10$, $i=5$ și $j=5$, se obțin la ieșire rezultatele $x=1$ și $y=1$, adică valori egale.

Răspuns corect: d)

Aplicație R.IV.4. Câtul și restul

(Bacalaureat iulie 2008, varianta 96)

Realizați un algoritm care citește de la tastatură două numere naturale distincte, apoi afișează câtul și restul împărțirii celui mai mare dintre cele două numere la cel mai mic dintre ele. Dacă numerele sunt egale, sau dacă împărțirea nu se poate efectua, se va afișa mesajul "Eroare".

Rezolvare

Pseudocodul acestui algoritm a fost explicat pe larg în cadrul aplicației R.II.4 din cadrul lecției "II.3.1 Structura de selecție simplă" (capitolul II).

Pseudocodul:

```

citește a,b;
dacă (a==b) atunci
    scrie "Eroare";
altfel
    început
        dacă (a>b) atunci
            dacă (b#0) atunci
                scrie [a/b], " ", a%b;
            altfel
                scrie "Eroare";
        dacă (b>a) atunci
            dacă (a#0) atunci
                scrie [b/a], " ", b%a;
            altfel
                scrie "Eroare";
    sfârșit

```

Programul:

```

#include <iostream.h>
#include <conio.h>

void main ()
{
    int a,b;

```

```

cout << "Dati numerele a si b:" << endl;
cin >> a >> b;
if (a==b)
    cout << endl << "Eroare";
else
{
    if (a>b)
        if (b) // echivalent cu "b!=0"
            cout << endl << "Catul si restul sunt " << a/b << " " << a%b;
        else
            cout << endl << "Eroare";
    if (b>a)
        if (a)
            cout << endl << "Catul si restul sunt " << b/a << " " << b%a;
        else
            cout << endl << "Eroare";
}
}

```

Încercați singuri!

►1. Scrieți programe pentru algoritmii propuși în tema T.4. din cadrul lecției "II.3.1 Structura de selecție simplă" (capitolul II).

Aplicație R.IV.5. Laturi în triunghi

Se citesc de la tastatură trei numere reale a , b , c . Să se verifice dacă aceste numere pot fi laturile unui triunghi și în caz afirmativ să se determine natura triunghiului în funcție de laturi (echilateral, isoscel, oarecare). Trei numere pot fi laturile unui triunghi, dacă sunt pozitive și suma a oricare două dintre ele este mai mare decât al treilea.

Exemplu: Numerele $a=4$, $b=4$ și $c=7$ sunt laturile unui triunghi isoscel, iar valorile $a=2$, $b=3$ și $c=9$ nu pot fi laturile unui triunghi (din cauză că $2+3<9$).

Rezolvare

Pseudocodul (aplicația R.II.5. din capitolul "Principiile programării structurate").

```

citește a,b,c;
dacă (a>0 și b>0 și c>0 și a+b>c și a+c>b și b+c>a) atunci
    început
        scrie "Numerele sunt laturi";
        dacă (a==b și b==c)
            scrie "Triunghi echilateral";
        altfel
            dacă (a==b SAU a==c SAU b==c)
                scrie "Triunghi isoscel";
            altfel
                scrie "Triunghi oarecare";
    sfârșit

```

```
altfel
    scrie "Nu e triunghi";
```

 Programul:

```
#include <iostream.h>

void main ()
{
    float a,b,c;
    float p,s;
    //citeste laturile in variabilele a, b, c
    cout << "\n Dati laturile: ";
    cin >> a >> b >> c;
    //verifica daca a, b, c, citite anterior, pot fi laturile unui triunghi
    if (a>0 && b>0 && c>0 && a+b>c && a+c>b && b+c>a)
    {
        cout << "\n Numerele sunt laturi";
        //verifica natura triunghiului
        if (a==b && b==c)
            cout << "\n triunghi echilateral ";
        else
            if (a==b || a==c || b==c)
                cout << "\n triunghi isoscel";
            else
                cout << "\n triunghi oarecare";
    }
    else
        cout << "\n Numerele nu pot fi laturi in triunghi";
}
```

 Încercați singuri !

►1. Scrieți programe pentru algoritmi propuși în tema T.5. din cadrul lecției "II.3.1 Structura de selecție simplă" (capitolul II).

 Aplicație R.IV.6. Rezolvarea ecuației de gradul II

Scrieți un program pentru rezolvarea ecuației de gradul II de forma $ax^2+bx+c=0$, unde coeficienții a, b, c se citesc de la tastatură. Se vor trata toate cazurile posibile.

 Rezolvare

 Pseudocodul (aplicația R.II.6. din capitolul "Principiile programării structurate").

```
citește a,b,c;
dacă (a==0) atunci
    dacă (b==0) atunci
        scrie "Nu se rezolva ecuatia"
```

```
altfel
    scrie -c/b
altfel
    inceput
        delta ← b*b-4*a*c;
        dacă (delta<0) atunci
            scrie "Nu are solutii reale"
        altfel
            inceput
                x1 ← (-b+√delta)/(2*a);
                x2 ← (-b-√delta)/(2*a);
                scrie x1,x2;
            sfârșit
    sfârșit
```

 Programul:

```
#include <iostream.h>
#include <math.h>

void main ()
{
    float a,b,c; float delta,x1,x2;
    cout << "\n Dati coeficientii: ";
    cin >> a >> b >> c; // citeste coeficientii
    if (a==0)
        if (b==0) // rezolva ec. de gr.I
            cout << "\n Nu se rezolva ecuatia";
        else
            cout << "\n Ec. de gr. I cu solutia " << (-c/b);
    else
    {
        delta=b*b-4*a*c; // discriminantul
        if (delta<0)
            cout << "\n Ecuatia nu are solutii reale!";
        else // cazul delta>=0
        { // rezolva ec. de gr.II
            x1=(-b+sqrt(delta))/(2*a);
            x2=(-b-sqrt(delta))/(2*a);
            cout << "\n Solutiile sunt:";
            cout << "x1=" << x1 << " x2=" << x2;
        }
    }
}
```

 Încercați singuri !

►1. Scrieți programe pentru algoritmi propuși în tema T.6. din cadrul lecției "II.3.1 Structura de selecție simplă" (capitolul II).

Încercați singuri!

►1. Transcrieți în limbajul de programare C++ toate pseudocodurile prezentate în enunțurile problemelor din tema T.2., din cadrul lecției "II.3.1 Structura de selecție simplă" (capitolul II).

Ați înțeles?

Probleme cu răspuns scurt

Următoarele două probleme se referă la secvența de program de mai jos:

```
if (a>0)
if (b>0) x=8;
else x=5;
else x=6;
cout << x;
```

►1. În urma execuției secvenței, pentru care dintre perechile de valori ale variabilelor întregi *a* și *b* date mai jos se va afișa *x=5*?

- | | |
|-------------------------------|------------------------------|
| a) <i>a=-1</i> și <i>b=-1</i> | b) <i>a=1</i> și <i>b=1</i> |
| d) <i>a=0</i> și <i>b=-1</i> | d) <i>a=1</i> și <i>b=-1</i> |

►2 Este posibil să găsim două valori egale între ele pentru variabilele *a* și *b*, astfel încât secvența să afișeze o valoare negativă?

►3. (Bacalaureat august 2001, varianta 2)

Știind că variabilele întregi *x* și *y* au inițial aceeași valoare, stabiliți care este diferența absolută între valorile lor la sfârșitul execuției secvenței următoare.

```
y++; x--;
if (y<=x)
{ y--; x++; }
else
{ y+=2; x--; };
```

►4. Se consideră secvența de mai jos, în care *x* este o variabilă de tipul caracter, iar *m* este de tip întreg.

```
if (x=='a') m=0; else if ((x=='b') || (x=='c')) m=1; else
if ((x>'c') && (x<'z')) m=0; else if (x=='z') m=1; else m=-1;
```

Pentru fiecare dintre secvențele S1, S2 și S3 date mai jos, stabiliți dacă este echivalentă cu secvența de mai sus. Justificați răspunsurile.

```
// Secvența S1
if (x=='a') m=0;
else
if (x=='b' || x=='c')
m=1;
else
if (x>'c' && x<'z')
m=0;
else
if (x=='z') m=1;
else m=-1;
```

```
// Secvența S2
if (x=='b' || x=='c' || x=='z')
m=1;
else
if ((x=='a') || (x>'c' && x<'z'))
m=0;
else
m=-1;
```

```
// Secvența S3
if (x<'a' || x>'z') m=-1;
else
if (x=='b' || x=='c' || x=='z')
m=1;
else
m=0;
```

►5. Precizați ce afișează programul de mai jos:

```
int a=1, b=0, c=1;
if (a || b) if (b && !c) cout << "unu"; else cout << "doi";
```

- | | | |
|---|----------------|----------------|
| a) Nu afișează nimic. | b) Textul unu. | c) Textul doi. |
| d) Instrucțiunea if este greșită sintactic. | | |

►6. Care dintre următoarele secvențe de instrucțiuni atribuie variabilei întregi *x* cea mai mică dintre valorile variabilelor întregi *a* și *b*, sau valoarea lor comună în cazul în care acestea sunt egale?

- | | |
|--------------------|-------------------|
| a) if (a==b) x=b; | b) x=a; |
| else | if (x>b) |
| if (b>a) x=a; | x=b; |
| c) if (b==a) x==a; | d) if (a<=b) x=b; |
| else x=b; | else x=a; |

►7. Fie *x* o variabilă de tipul float. Precizați care dintre secvențele Pascal de mai jos corespunde enunțului "dacă valoarea lui *x* este un număr întreg (fără parte zecimală), atunci afișează TRUE".

- | |
|--------------------------------------|
| a) if (x==(int)x) cout << 1; |
| b) if (x==floor(x)) cout << 1; |
| c) if (x==ceil(x)) cout << 1; |
| d) if (floor(x)==ceil(x)) cout << 1; |

►8. Precizați de câte ori se va afișa valoarea 1 în timpul execuției programului următor, dacă prin citire de la tastatură variabilele primesc valorile *a=3*, *b=4* și *x=5*.

```
#include<iostream.h>
void main()
{
int a,b,x;
cin >> a >> b >> x;
if (!(x<=a && (x>=b))) cout << 1 << "\n";
```

```

if (!(x<=a || x>=b)) cout << 1 << "\n";
if (!(x<=a) && !(x>=b)) cout << 1 << "\n";
if (!(x<=a) || !(x>=b)) cout << 1 << "\n";
}

```

Următorii trei itemi se referă la secvența de program de mai jos:

```

int x=floor(sqrt(10));
int a,b,c;
a=b=c=0;
if (x>3)
a++;
else
if (x<-3);
b++;
else
if (x<0)
c++;
cout << "\n" << a << b << c;

```

►9. Secvența dată conține erori de sintaxă? În caz afirmativ, câte și care sunt acestea?

►10. După corectarea erorilor, în urma execuției secvenței, care dintre variabilele întregi a, b, c va primi valoarea 1?

- a) variabila a b) variabila b c) variabila c
d) niciuna dintre cele trei variabile

►11. Înlocuiți structura de selecție "if (x<0) else c++;" cu una echivalentă (cu același efect), dar care să nu mai conțină ramura "else".

►12. (Bacalaureat iulie 2009, varianta 96)

În secvența dată de instrucțiuni, variabilele i, j, k și y sunt de tip întreg. Pentru care dintre următoarele seturi de valori ale variabilelor i, j și k variabila y va avea valoarea 1 în urma executării secvenței?

```

if (k>0)
if (i!=j) y=0;
else y=1;
else y=2;

```

- a) k=0; i=5; j=5; b) k=10; i=5; j=6;
c) k=10; i=5; j=5; d) k=0; i=5; j=6;

►13. (Bacalaureat iulie 2009, varianta 1)

Știind că variabilele a și b sunt utilizate pentru a memora două numere reale, stabiliți care dintre secvențele de instrucțiuni de mai jos determină, în urma executării, inițializarea variabilei m cu diferența absolută a valorilor memorate în a și b.

- a) m=(a-b) || (b-a);
b) m=a; if (m<a) m=b-m;
c) m=a; if (m>b) m=a-m;
d) m=a-b; if (a<b) m=-m;

►14. (Bacalaureat iulie 2008, varianta 61)

Deduceți care este cea mai mică valoare pozitivă pe care o poate memora

variabila x, astfel încât, în urma executării secvenței de instrucțiuni de mai jos, să se afișeze valoarea lui x.

```

if (x>71%2+3/x) then
cout << x;

```

- a) 2 b) 3 c) 4 d) 5



Probleme propuse

Scrieți programe pentru algoritmi propuși în tema T.R.1. din lecția "II.3.1. Structura de selecție simplă" (capitolul II), la sfârșitul secțiunii "Aprofundare".



IV.1.2. Instrucțiunea de selecție multiplă

Instrucțiunea de selecție multiplă case este implementarea structurii de selecție multiplă descrisă în pseudocod. Ea permite alegerea unei secvențe de instrucțiuni din mai multe posibile, în funcție de valoarea unei expresii numită selector. De cele mai multe ori selectorul este o variabilă.

```

alege (<selector>)
început
    cazul <val_1>:
        <secvența_1>
        ieșire
    cazul <val_2>:
        <secvența_2>
        ieșire
    .....
    cazul <val_n>:
        <secvența_n>
        ieșire
altfel
    <secvența_0>
sfârșit

```

```

switch (<selector>)
{
    case <val_1>:
        <secvența_1>
        break;
    case <val_2>:
        <secvența_2>
        break;
    .....
    case <val_n>:
        <secvența_n>
        break;
    default
        <secvența_0>
}

```

Așadar:

- Linia "switch (<selector>)" se traduce prin "alege unul din mai multe cazuri posibile, în funcție de valoarea variabilei (<selector>". În limbajul C++, numele variabilei selector trebuie scris între paranteze rotunde.

- Fiecare dintre simbolurile $\langle val_1 \rangle$, $\langle val_2 \rangle$, ... $\langle val_n \rangle$, reprezintă o valoare posibilă a selectorului $\langle selector \rangle$. Astfel, de exemplu, linia "case $\langle val_1 \rangle$:" se traduce prin "cazul în care selectorul are valoarea simbolizată prin $\langle val_1 \rangle$ ";
- Fiecărui caz îi corespunde o secvență de instrucțiuni, iar calculatorul va alege unul dintre cazuri executând secvența aferentă în funcție de valoarea selectorului. Astfel, în cazul în care selectorul are valoarea $\langle val_1 \rangle$ se execută secvența de instrucțiuni $\langle secvența_1 \rangle$, dacă selectorul are valoarea $\langle val_2 \rangle$ are loc execuția secvenței $\langle secvența_2 \rangle$, ș.a.m.d.. Observați că pe fiecare "ramură", după secvența de instrucțiuni aferentă, urmează cuvântul cheie "break;". Acest fapt se datorează modului în care funcționează instrucțiunea de selecție multiplă în limbajul C++: în absența instrucțiunii "break", compilatorul ar trece la analizarea următoarelor "cazuri". Prezența cuvântului-cheie "break" la sfârșitul fiecăruia dintre "cazuri", va determina compilatorul să abandoneze complet instrucțiunea de selecție multiplă după execuția secvenței de instrucțiuni aferentă "cazului" selectat.
- Pe lângă "ramurile" ce descriu cazurile, mai poate exista o ramură numită "default". Aceasta se va executa dacă valoarea selectorului nu este cuprinsă în nici unul dintre cazurile anterioare $\langle val_1 \rangle$, $\langle val_2 \rangle$, ..., $\langle val_n \rangle$. Menționăm însă că prezența ramurii "default" nu este obligatorie, situație în care dacă nu este îndeplinit nici unul din cazurile date, pur și simplu nu se va executa nimic.
- Întreaga instrucțiune de selecție multiplă trebuie cuprinsă între acoladă deschisă și închisă.

Exemplu:

Prezentăm un program pur demonstrativ care, fără a avea vreun efect deosebit, ilustrează sintaxa instrucțiunii case. Pseudocodul acestui program a fost explicat pe larg în capitolul "Algoritmi. Principiile programării structurate", lecția "Structura de selecție multiplă".

```
#include<iostream.h>
int x,y;
void main()
{
    cout << "x=";
    cin >> x;
    switch (x)
    {
        case -1:
            y=x+2;
            break;
        case 0:
            cout << "Ati introdus 0";
            y=x;
            break;
        case 1:
            y=x-1;
```

```
citește x;
alege (x)
început
    cazul -1:
        y←x+2;
        ieșire
    cazul 0:
        scrie "Ati introdus 0";
        y←x;
        ieșire
    cazul 1:
        y←x-1;
        ieșire
    altfel
        scrie "Nu e -1,0,1 ";
        y←100;
sfârșit
```

```
break;
default:
    cout << "Nu e -1,0,1";
    y=100;
}
cout << "\n" << y;
}
```

Aplicație R. IV.7. Grilă

Ce va afișa programul următor, dacă de la tastatură se introduc în ordine numerele 5, 6 și 8 ?

```
#include <iostream.h>
void main()
{
    int x,y,z,m;
    cin >> x >> y >> z;
    m=(x+y+z)/3;
    switch (m)
    {
        case 4:
            cout << "Corigent";
            break;
        case 6:
            cout << "Mediocru";
            break;
        case 8:
            cout << "Bine";
            break;
        case 10:
            cout << "Foarte bine";
            break;
        default:
            cout << "EROARE !";
    }
}
```

a) "Corigent" b) "Mediocru" c) "Satisfacator" d) "Foarte bine"

Rezolvare

 Acțiunile programului sunt următoarele:

- se citesc de la tastatură valorile variabilelor întregi x , y și z ;
- se atribuie variabilei m valoarea expresiei $(x+y+z) / 3$. Dacă se citesc $x=5$, $y=6$, $z=8$, atunci $m=(5+6+8) / 3=19 / 3=6$ (reamintim că atunci când deîmpărțitul și împărțitorul sunt ambii întregi, operatorul "/" furnizează câtul împărțirii întregi);

• instrucțiunea switch începe cu linia de program "switch (m)". Aceasta înseamnă că alegerea uneia sau alteia dintre ramurile instrucțiunii se va face în funcție de valoarea variabilei m . Cazurile posibile sunt:

- ✓ (C1) $m=4$; (C2) $m=6$; (C3) $m=8$;
- ✓ (C4) $m=10$; (C5) default $\Rightarrow m$ are orice valoare în afara celor de mai sus.

 Deoarece variabila m a primit anterior valoarea 6, rezultă că este îndeplinit cazul (C2), deci se va executa instrucțiunea aferentă acestuia: {cout << "Mediocru";}

Așadar programul afișează șirul de caractere "Mediocru".

Răspuns corect: b).

Aplicație R.IV.8. Alegere

Să se alcătuiască un program care citește două numere reale a și b , apoi afișează media aritmetică, suma pătratelor sau suma cuburilor celor două numere, în funcție de dorința utilizatorului.

Rezolvare

Pseudocodul (Aplicația R.II.6. din capitolul "Principiile programării structurate").

```
citește a,b;
citește opt;
alege (opt)
început
    cazul 1:
        scrie (a+b)/2;
        ieșire
    cazul 2:
        scrie a*a+b*b;
        ieșire
    cazul 3:
        scrie a*a*a+b*b*b;
        ieșire
    altfel
        scrie "Eroare";
sfârșit
```

Programul:

```
#include <iostream.h>

void main ()
{
    float a,b,ma,mb;
    int opt;
    // citește numerele. a, b și selectorul opt; valorile "valide" ale lui opt vor fi 1,2,3
    cout << "\n Introduceți numerele : ";
    cin >> a >> b;
    cout << "\n Ce doriți să calculez ?";
    cout << "\n Dati opțiunea dvs, tastând o cifră:";
    cout << "\n 1: media aritmetică";
    cout << "\n 2: suma patratelor";
    cout << "\n 3: suma cuburilor \n";
    cin >> opt;
    switch (opt) // alege în funcție de valoarea selectorului opt
    {
        case 1: // dacă opt=1, calculează media aritmetică, afișând-o direct
            cout << "Media aritm=" << (a+b)/2;
            break;
        case 2: // dacă opt=2, calculează suma patratelor, afișând-o direct
            cout << "\n Suma patratelor=" << a*a+b*b;
            break;
```

```
case 3: // dacă opt=3, calculează suma cuburilor, afișând-o direct
    cout << "\n Suma cuburilor=" << a*a*a+b*b*b;
    break;
default: // pentru orice altă valoare a lui opt, în afara de 1,2,3, nu execută nimic
    cout << "\n Opțiune incorectă ! Abandonez !";
} // de la switch
}
```

Încercați singuri !

Scrieți un program pentru algoritmul propus în tema T.7. din lecția "II.3.2. Structura de selecție multiplă" (capitolul II).

Ați înțeles ? Probleme cu răspuns scurt

►1. Fie secvența de program alăturată. Pentru care dintre perechile de valori ale variabilelor întregi x și y date mai jos, secvența afișează de două ori valoarea 0 ?

- a) $x=-4, y=5$
- b) $x=-1, y=3$
- c) $x=0, y=0$
- d) $x=0, y=1$
- e) $x=6, y=-2$

►2. În programul alăturat, ce atribuire poate fi scrisă în loc de "...", astfel încât să se afișeze 0 ?

- a) $a=6$;
- b) $a=7$;
- c) $a=8$;
- d) $a=9$;

►3. Dacă în timpul execuției programului de mai jos n va primi prin citire de la tastatură valoarea 232213, care vor fi în final valorile variabilelor $f1$, $f2$ și $f3$?

```
#include <iostream.h>
void main()
{
    switch (c) {
        case 1: { f1++; break; }
```

```
#include <iostream.h>
int x,y;
void main()
{
    cin >> x >> y;
    switch (x+y)
    {
        case 1: y=y-1; break;
        case 2: x=0; y=0; break;
        case 0: x++; break;
        default: break;
    }
    cout << "x=" << x << "\n y=" << y;
}
```

```
#include <iostream.h>
int a,b;
void main()
{
    b=1;
    cout << "Introduceți a=";
    cin >> a;
    switch (a)
    {
        case 6: b=b && 1; break;
        case 7: b=b && 0; break;
        case 8: b=b || 1; break;
        case 9: b=b || 0; break;
        default: break;
    }
    cout << "b=" << b;
}
```



```

long n;
unsigned int f1,f2,f3,c;
cin >> n;
f1=f2=f3=0;
do {
    c=n%10;
    n=n/10;
    case 2: { f2++; break;}
    case 3: { f3++; break;}
} while (n!=0);
cout << f1 << f2 << f3;
}

```

- a) f1=1, f2=1, f3=1 c) f1=1, f2=3, f3=2 e) f1=3, f2=2, f3=1
b) f1=1, f2=2, f3=2 d) f1=2, f2=1, f3=3



Probleme propuse

Scrieți programe pentru algoritmi propuși în tema T.7. din lecția "2.3.2. Structura de selecție multiplă" (capitolul II).



IV.2. Instrucțiuni repetitive (cicluri)

IV.2.1. Ciclul cu test inițial (while-do)

Reprezintă codificarea în limbajul C++ a structurii repetitive cu test inițial pe care am descris-o în capitolul "Algoritmi. Principiile programării structurate".

```

cât timp (<expr>) execută
<secv>          while (<expr>)
                  <secv>

```

– unde <expr> este o *condiție (expresie logică)*, iar <secv> este o secvență formată dintr-una sau mai multe instrucțiuni, numită *corpul ciclului*.

Principiul de funcționare:

Atâta timp cât este îndeplinită condiția dată de <expr> (cât timp expresia <expr> are valoarea 1), execută corpul ciclului <secv>:

- 1) se evaluează expresia logică <expr>
 - 2) – dacă aceasta este adevărată, atunci se execută secvența <secv>, apoi se revine la pasul "1" (între timp valoarea expresiei <expr> s-a putut modifica);
- în caz contrar se trece la prima instrucțiune de după ciclu.

Observații:

- Expresia logică <expr> va fi scrisă întotdeauna între paranteze rotunde;
- În cazul în care corpul ciclului conține cel puțin două instrucțiuni, acesta va fi cuprins între acoladă deschisă și închisă;
- Testarea condiției are loc la început, deci corpul ciclului poate să nu se execute niciodată.

Exemplu:

Să se calculeze suma a n numere naturale citite pe rând de la tastatură (unde n este cunoscut).

Pseudocodul acestei probleme a fost explicat pe larg în lecția "Structura repetitivă cu test inițial" din capitolul "Principiile programării structurate". Reamintim pe scurt algoritmul problemei. La fiecare execuție a corpului ciclului se citește un număr în variabila x și se adaugă la S, apoi se incrementează contorul i care ne spune la a câta execuție ne aflăm. Variabila i se inițializează cu 1, iar ciclul se execută *cât timp* $i \leq n$ (se citesc n numere).

```
#include <iostream.h>
```

```

void main ()
{
    int i,n,x,S;
    cout << "n=";
    cin >> n;
    S=0;
    i=1;
    while (i<=n)
    {
        cout << "x="; cin >> x;
        S+=x; i++;
    }
    cout << "S=" << S;
}

```

```

citește n;
S←0;
i←1;
cât timp (i<=n) execută
    început
        citește x;
        S←S+x;
        i←i+1;
    sfârșit
scrie S;

```

Algoritmi cu structuri repetitive cu test inițial

Aplicație

R. IV.9. Problemă cu răspuns scurt

Știind că toate variabilele sunt de tip întreg, care dintre secvențele următoare afișează 12?

- a)

```
i=1;
while (i<=2)
{
    cout << i;
    i++;
}
```
- b)

```
i=1; S=7;
while (i<=2)
    i++; S+=i;
cout << S;
```
- c)

```
a=1;
i=floor(0.2);
j=abs(-3);
while (a)
{
    i++; j--;
    a=(i>j);
};
cout << i << j;
```
- d) nici una dintre secvențele date.

Rezolvare

Analizăm pe rând secvențele date.

Secvența a):

Contorul i se inițializează cu 1 și cât timp $i \leq 2$ execută două instrucțiuni: tipărește valoarea lui i , apoi incrementează i cu 1. Pașii ciclului sunt următorii:

Pas 1: $i \leq 2$? $1 \leq 2$? da $\Rightarrow$ scrie 1 și $i = i + 1$, $i = 1 + 1 = 2$

Pas 2: $i \leq 2$? $2 \leq 2$? da $\Rightarrow$ scrie 2 și $i = i + 1$, $i = 2 + 1 = 3$

Pas 3: $i \leq 2$? $3 \leq 2$? nu $\Rightarrow$ iese din ciclu

Prin urmare, s-au afișat cifrele 1 și 2 "lipite" una de alta, simulându-se numărul întreg 12 (enunțul nu cere să se formeze în memorie numărul 12 !).

Secvența b):

Cât timp $i \leq 2$, se execută incrementarea $i = i + 1$. Atenție, instrucțiunea $S = S + i$, adică $S = S + i$, nu este în ciclu ! În absența acoladelor deschisă și închisă, corpul ciclului este considerat numai prima instrucțiune de după linia "while". Valoarea inițială a lui i este 1, deci după ieșirea din ciclu vom avea $i = 3$ (ciclul are doi pași, iar la finele celui de-al doilea incrementarea $i = i + 1$ va conduce la $i = 2 + 1 = 3$). Apoi, după ciclu, atribuirea $S = S + i$ va avea ca rezultat $S = 7 + 3 = 10$.

Secvența c):

Reamintim că funcția `floor(0.2)` trunchiază pe 0.2 la cel mai apropiat întreg mai mic decât 0.2, deci returnează 0, iar funcția `abs(-3)` întoarce $|-3|$ adică 3. Valorile returnate de către cele două funcții se atribuie variabilelor i respectiv j . Prin urmare valorile inițiale sunt $a = 1$, $i = 0$ și $j = 3$.

Cât timp a este diferit de 0 (condiție care este scrisă simplificat "while (a)" în loc de "while (a != 0)"), execută trei atribuiri. Ultima dintre ele, " $a = (i > j)$ ", atribuie variabilei a valoarea de adevăr a expresiei logice $i > j$. La primul pas al ciclului avem $i = i + 1 = 1$, $j = j - 1 = 2$ și $a = 0$ (expresia " $i > j$ " este falsă), apoi condiția " $a != 0$ " din linia while este deja falsă producând ieșirea din ciclu. În final, se afișează valorile variabilelor i și j una după alta, fără nici un separator între ele, tipărindu-se astfel 12.

Răspuns corect: a) și c).

Sume și produse cu n termeni

Aplicație R.IV.10. Produs de numere naturale impare

Scrieți pseudocodul unui algoritm care calculează produsul numerelor naturale impare mai mici sau egale decât o valoare n dată.

Exemplu: pentru $n = 8$ rezultă $P = 1 \cdot 3 \cdot 5 \cdot 7 = 105$, pentru $n = 9$ se va calcula $P = 1 \cdot 3 \cdot 5 \cdot 7 \cdot 9 = 945$.

Rezolvare

Algoritmul a fost explicat în aplicația R.II.10. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Pseudocodul:

```
citește n;
P ← 1; i ← 1;
cât timp (i ≤ n) execută
    început
        P ← P * i; i ← i + 2;
    sfârșit
scrie P;
```

Programul:

```
#include <iostream.h>

void main ()
{
    int P, i, n;
    // citește de la tastatură valoarea lui n
    cout << "\n n=";
    cin >> n;
    // calculează produsul cerut în variabila P
    P = 1; i = 1;
    while (i ≤ n)
    {
        P *= i;
        i += 2;
    }
    cout << "\n P=" << P;
}
```

Încercați singuri !

Scrieți programe pentru algoritmi propuși în tema T.10. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Șiruri de numere citite pe rând

Aplicație R.IV.11. Numărare într-un șir de numere a cărui citire se termină cu 0

Se dă un șir de numere care se citesc pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din șir). Să se afișeze câte numere pozitive sunt în șir.

Rezolvare

Algoritmul a fost explicat în aplicația R.II.11. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

```
#include <iostream.h>

void main ()
{
    int x,nr;
    cout << "\n x=";
    cin >>x;
    nr=0;
    while (x)
    {
        if (x>0)
            nr++;
        cout << "\n x=";
        cin >>x;
    }
    cout << "\n nr=" << nr;
}
```

```
citește x;
nr←0;
cât timp (x≠0) execută
    început
    dacă (x>0) atunci
        nr←nr+1;
    citește x;
    sfârșit
    scrie nr;
```

Încercați singuri!

Scrieți programe pentru algoritmi propuși în tema T.11. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Aplicație R.IV.12. Maximul într-un șir de numere a cărui citire se termină cu 0

Se dă un șir de numere care se citesc pe rând de la tastatură, atâta timp cât nu s-a introdus valoarea 0 (care nu face parte din șir). Să se afișeze maximul dintre elementele șirului.

Rezolvare

Algoritmul a fost explicat în aplicația R.II.10. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

```
citește x;
max←-x;
cât timp (x≠0) execută
    început
    dacă (x>max) atunci
        max←x
    citește x;
    sfârșit
    scrie max
```

Programul:

```
#include <iostream.h>

void main ()
{
    int x,max;
```

```
cout << "\n x="; cin >> x;
max=x;
while (x!=0)
{
    if (x>max)
        max=x;
    cout << "\n x="; cin >> x;
}
cout << "\n Maximul este " << max;
}
```

Încercați singuri!

Scrieți programe pentru algoritmi propuși în tema T.12. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Prelucrarea cifrelor unui număr natural

Aplicație R.IV.13. Suma cifrelor unui număr

Realizați un algoritm care calculează suma cifrelor unui număr natural x citit de la tastatură.

Rezolvare

Algoritmul a fost explicat în aplicația R.II.13. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

```
citește x;
d←x;
S←0;
cât timp (d≠0) execută
    început
    S←S+d % 10;
    d←d / 10;
    sfârșit
    scrie S;
```

Programul:

```
#include <iostream.h>

void main()
{
    int S;
    long d,x;
    // citește de la tastatură numărul x
    cout << "\n x=";
    cin >> x;
    // inițial deimpărțitul d este chiar numărul x, iar suma cifrelor S este 0
    d=x;
```

```

S=0;
// extrage cifrele lui x intr-un ciclu, adaugand la S fiecare cifra
while (d) // cat timp deimpartitul este diferit de 0
{
    S+=d % 10; // adauga cifra extrasa d % 10 la suma cifrelor S
    d=d/10; // catul impartirii curente, d / 10, va fi deimpartit pentru impartirea urmatoare
}
cout << "\n S=" << S;
}

```

Încercați singuri !

- 1. Scrieți programe pentru algoritmi propuși în tema T.13. din lecția "2.4.1. Structura repetitivă cu test inițial" (capitolul II).
- 2. Scrieți un program pentru algoritmi propuși în tema T.14. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Interpretarea unui pseudocod dat

Încercați singuri !

Scrieți programe Pascal pentru toți algoritmi dați în pseudocod în enunțurile problemelor din temele T.15 și T.16.

Algoritmi cu divizori și factori primi

Aplicație R.IV.14. Frație zecimală

(Bacalaureat iulie 2009, varianta 25, enunț adaptat)

Se citește de la tastatură un număr real pozitiv x , care are cel mult două cifre în partea întreagă și cel mult șapte cifre după punctul zecimal. Scrieți un program care, utilizând un algoritm eficient din punctul de vedere al memoriei utilizate și al timpului de executare, afișează pe ecran, separate printr-un spațiu, două numere naturale al căror raport este egal cu x , și a căror diferență în modul este minimă.

Exemplu: dacă se citește $x=0.375$, se vor afișa pe ecran numerele 3 și 8 (deoarece $3/8=0.375$, iar expresia $|8-3|$ este cea mai mică dintre toate diferențele în modul dintre numitorii și numărătorii fracțiilor prin care s-ar putea scrie valoarea zecimală 0.375).

Rezolvare

Algoritmul acestei probleme a fost explicat pe larg în cadrul aplicației R.II.17. din cadrul lecției "II.4.1. Structura repetitivă cu test inițial".

Pseudocodul:

```

citește x;
q←1;
cât timp (x#[x]) execută
    început
        q←q*10;
        x←x*10;
    sfârșit
p←x;
scrie p,q;
a←p;
b←q;
cât timp (a#b) execută
    dacă (a>b) atunci
        a←a-b
    altfel
        b←b-a;
d←a;
p←p/d;
q←q/d;
scrie p,q;

```

Programul:

```

// Bac 2008, II.4 / var 25
#include <iostream.h>
#include <conio.h>
#include <math.h>

void main ()
{
    clrscr();
    float x;
    int a,b,d,p,q;
    // citește numărul real x
    cout << "Dati numărul x: "; cin >> x;
    // p si q vor fi numărătorul si numitorul fracției prin care poate fi scris numărul real x
    // in mod repetat inmultim pe x cu 10, pana cand el nu mai contine zecimale
    q=1;
    // prin variabila q tinem minte cate inmultiri facem
    // iar valoarea finala a lui q va fi numitorul fracției
    while (x!=floor(x))
    { q=q*10; x=x*10; }
    p=x; // valoarea finala a lui x va fi numărătorul fracției
    cout << " numarat si numit fracției sunt " << p << " si " << q;
}

```

```
// aflam c.m.m.d.c al lui p si q, fie acesta d
a=p;
b=q;
while (a!=b)
    if (a>b) a=a-b;
    else b=b-a;
d=a;
// simplificam la maximum posibil fractia p/q, prin c.m.m.d.c. al lui p si q
p=p/d; q=q/d;
cout << "\n numarat si numit fractiei sunt " << p << " si " << q;
getch();
}
```

Încercați singuri!

Scrieți un program pentru algoritmi propuși în tema T.17. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Aplicație R.IV.15. Număr prim

Scrieți un algoritm care verifică dacă un număr natural x diferit de 0 și 1, este prim sau nu, afișând un mesaj corespunzător.

Rezolvare

Algoritmul a fost explicat în aplicația R.II.18 din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II). Am prezentat atunci doua varietate de pseudocod: prima descrie algoritmul "clasic", iar a doua aduce niște îmbunătățiri care vizează eliminarea unor operații inutile. Prezentăm și programele corespunzătoare pentru ambele variante.

Variantă A:

```
citește x;
ok←1;
i←2;
cât timp (i≤ x / 2 și ok=1) execută
    început
        dacă (x % i=0) atunci
            ok←0;
            i←i+1;
        sfârșit
    dacă (ok=1) atunci
        scrie "Numărul este prim";
    altfel
        scrie "Numărul NU este prim";
```

Programul:

```
#include <iostream.h>
void main ()
```

```
{
    int x,i,ok;
    // citește numărul x de la tastatura
    cout << "\n x="; cin >> x;
    // variabila ok va memora "starea" condiției testate: TRUE dacă x este prim,
    // respectiv FALSE în caz contrar; inițial, presupunem ca x este prim
    ok=1; i=2; // prin variabila i vor "trece" posibili divizori ai lui x, 2,3,...,x div 2
    // cât timp nu am parcurs toți divizorii posibili și nu am găsit încă un divizor
    while (i≤x/2 și ok)
    {
        // dacă valoarea curentă a lui x este divizor al lui x, atunci x nu mai poate fi prim
        if (x%i==0)
            ok=0;
        i++; // trecem la următorul posibil divizor i
    }
    // în final, testăm valoarea variabilei ok, afișând un mesaj corespunzător
    if (ok)
        cout << "\n Numărul este prim";
    else
        cout << "\n Numărul nu este prim";
}
```

Variantă B:

```
citește x;
dacă (x % 2=0)
    dacă (x=2)
        ok←1
    altfel
        ok←0
    altfel
        început
            i←2;
            cât timp (i≤x/2 și ok=1)
                început
                    dacă (x % i=0)
                        ok←0;
                        i←i+1;
                    sfârșit
            sfârșit
        dacă (ok=1)
            scrie "Numărul este prim";
        altfel
            scrie "Numărul NU este prim";
```

Programul:

```
#include <iostream.h>
void main ()
{
    int x,i,ok;
    // citește numărul x de la tastatura
    cout << "\n x=";
    cin >> x;
```

```
// numerele pare nu pot fi prime, cu exceptia lui 2 care este considerat prim
if (x % 2==0)
    if (x==2) ok=1;
    else ok=0;
else
{
    // daca numarul x este impar, se aplica algoritmul de cautare a divizorilor din var. anterioara
    ok=1; i=2;
    while (i<=x/2 && ok)
    {
        if (x%i==0)
            ok=0;
        i++;
    }
}
if (ok)
    cout << "\n Numarul este prim";
else
    cout << "\n Numarul nu este prim";
}
```

Încercați singuri !

Scrieți un program pentru algoritmi propuși în tema T.18. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Ați înțeles ?

Probleme cu răspuns scurt

► 1. (Bacalaureat iulie 2008, varianta 6)

Pentru care dintre următoarele valori ale variabilei n , secvența de program de mai jos afișează valoarea 0 în urma executării ei ?

```
while (n%10>=2)
    n/=10;
cout << n;
```

- a) 1111 b) 9282 c) 3003 d) 1345

► 2. (Bacalaureat iulie 2008, varianta 9)

Deduceți valorile pe care le vor primi variabilele x și y , după execuția secvenței de program de mai jos:

```
x=1;
y=11;
while (x<=y)
{
    x++;
    y--;
}
```

► 3. Pentru $n=7$, care dintre secvențele de program de mai jos trebuie executată astfel încât, la finele execuției, valoarea variabilei P să fie 48 ?

a)

```
P=1; i=2;
while (i<=n)
{
    P*=i;
    i+=2;
}
```

c)

```
P=i=1;
while (i<=n/2)
{
    P=P*(2*i);
    i++;
}
```

b)

```
P=i=1;
while (i < n/2)
{
    i++;
    P=P*(2*i+1);
}
```

d)

```
P=1; i=-1;
while (i<n)
{
    i+=2;
    P*=i;
}
```

► 4. Secvența de program următoare afișează valoarea

```
#include <iostream.h>
void main()
{
    int n=1008,m=416;
    while (n!=m)
        if (n<m)
            m=m%n;
        else
            n=n%m;
    cout << n;
}
```

- a) 416
b) 0
c) 36
d) programul este greșit, deoarece produce o împărțire la 0

► 5. (Bacalaureat iulie 2001, varianta 2)

Știind că x și y desemnează variabile întregi, determinați valoarea inițială a variabilei x astfel încât secvența dată să afișeze exact un asterisc (*).

- a) -4 b) -2 c) 0 d) 1
e) 2 f) 4 g) 5

```
y=x;
while (x<=3)
{
    cout << "*";
    y++;
    x+=y;
}
```

► 6. (Bacalaureat august 2000, varianta 1)

Care dintre următoarele secvențe de instrucțiuni atribuie variabilei întregi u valoarea ultimei cifre a numărului natural reprezentat de variabila x ?

- a) $u=x/10$; b) $u=x$;
 while ($u>10$) $u=u/10$;
c) $u=x\%10$; d) while ($x>=10$) $x=x/10$;
 $u=x$;

► 7. Care dintre secvențele de program de mai jos afișează toate cifrele sistemului zecimal, în ordine descrescătoare de la 9 la 0 ?

{Secvența S1}

```
i=9;
while (i>=0)
{
    j=0;
```

{Secvența S2}

```
i=0;
while (i<9)
{
    j=1;
```

```

while (j<i)
    j++;
cout << j << " ";
i--;
}

```

- a) Numai secvența S1
c) Atât secvența S1 cât și S2

```

while (j<9-i)
    j++;
cout << j << " ";
i++;
}

```

- b) Numai secvența S2
d) Nici una dintre cele două secvențe

Următoarele patru probleme se referă la secvența de mai jos, descrisă în pseudocod:

```

cin >> n; q=1;
while (n>0)
{
    if (n%5==0) q=q*10;
    else q=q*10+1;
    n=n/5;
}
cout << q;

```

► 8. (Bacalaureat iulie 2009, varianta 58)

Deduceți valoarea afișată pentru $n=53$;

► 9. (Bacalaureat iulie 2009, varianta 58)

Găsiți un număr de două cifre care ar putea fi citit în variabila n , astfel încât programul să afișeze 1001.

► 10. Rescrieți instrucțiunea de selecție prezentă în cadrul secvenței anterioare, inversând ramurile "if" și "else" între ele.

► 11. Știind că toate variabilele din secvența anterioară pot memora doar numere naturale cu maxim patru cifre, scrieți declarațiile de variabile optime din punctul de vedere al spațiului de memorie rezervat.

► 12. (Bacalaureat iulie 2008, varianta 18)

Câte atribuiri au loc în total în timpul execuției secvenței de program de mai jos, în cazul în care variabila nu are valoarea 245?

```

s=0;
while (n)
{
    s++; n/=100;
}

```

► 13. Știind că a și b sunt numere întregi citite de la tastatură cu $a \geq b$, precizați ce secvență de instrucțiuni trebuie scrisă în loc de "..." în secvența următoare, așa încât valoarea variabilei r să reprezinte cel mai mare divizor comun al numerelor a și b .

```

r=0;
while (a % b)
{
    .....
}

```

- a) $a=b$; $b=r$; $r=a \% b$;
c) $r=a \% b$; $b=a$; $a=r$;

- b) $b=a$; $a=r$; $r=a \% b$;
d) $r=a \% b$; $a=b$; $b=r$;

► 14. (Bacalaureat, iulie 2002, varianta 3)

Care este valoarea inițială a variabilei întregi n astfel încât la sfârșitul execuției următoarei secvențe de instrucțiuni rezultatul afișat pe ecran să fie 503 ?

```

while (n>0)
{
    cout << (n % 10);
    n=n / 10-1;
}

```

- a) 305 b) 415 c) 425 d) 503

► 15. (Bacalaureat, iulie 2002, varianta 5)

Știind că valoarea inițială a variabilei întregi i este mai mare decât 10, stabiliți care este valoarea expresiei $\text{abs}(3-i)$ la sfârșitul executării următoarei instrucțiuni.

```
while (i>4) i--;
```

- a) -1 b) 0
c) 1 d) 2
e) o valoare nedeterminată

► 16. Ce valoare se afișează în urma execuției secvenței de mai jos ?

```

i=4; s=0;
while (i<=10)
    i++; s+=i;
cout << "n" << s;

```

- a) 0 b) 10 c) 49 d) 11

► 17. (Bacalaureat iulie 2008, varianta 20)

Care trebuie să fie valoarea inițială a variabilei n , astfel încât, la sfârșitul execuției secvenței de mai jos să se afișeze valoarea 3 ?

```

c=0;
while (n%10==0)
{
    n/=10;
    c++;
}

```

- a) 123
b) 10020
c) 5000
d) 10001

► 18. (Bacalaureat iulie 2008, varianta 67)

Pentru secvența de program de mai jos în care variabilele i și j sunt de tip întreg, alegeți o valoare inițială pentru variabila j , astfel încât executarea secvenței să se realizeze în timp finit.

```

i=0;
j=...;
while (i+j<=10)
{
    i++; j-=2;
}

```

- a) 6 b) 1 c) 5 d) 17

► 19. (Bacalaureat iulie 2008, varianta 39)

În secvența de program de mai jos, știind că de la tastatură se citește valoarea $n=234$, deduceți ce valoare se afișează după executarea secvenței.

```
cin >> n;
x=1;
while (n>0)
{
    x*=n%10; n/=10;
}
cout << x;
```



Probleme propuse

Folosind cicluri cu test inițial, scrieți programe pentru toți algoritmi propuși în tema T.R.3. de la sfârșitul capitolului II (secțiunea "Probleme recapitulative pentru toate tipurile de cicluri").



IV.2.2. Ciclu cu test final (do-while)

Implementează în limbajul Pascal structura repetitivă cu test final, care a fost de asemenea explicată în capitolul "Algoritmi. Principiile programării structurate".

```
do
    <secv>
while (<expr>);
```

– unde <expr> este o *condiție (expresie logică)*, iar <secv> este o secvență formată dintr-una sau mai multe instrucțiuni, numită *corpul ciclului*.

Principiul de funcționare:

Repetă execuția secvenței de instrucțiuni <secv>, **până când** expresia logică <expr> ia valoarea 1 (până când condiția aferentă devine adevărată).

- 1) se execută secvența de instrucțiuni <secv>;
- 2) se evaluează expresia logică <expr>;

- dacă condiția dată de <expr> este adevărată, adică dacă expresia <expr> are valoarea 1, atunci se revine la pasul "1)";
- în caz contrar, se iese din ciclu și se execută prima instrucțiune de după ciclu.

Observații:

- ✓ La fel ca și în cazul structurii repetitive cu test inițial, expresia <expr> din linia "cât timp" se va pune între paranteze rotunde
- ✓ În cazul în care secvența de instrucțiuni <secv> ce reprezintă corpul ciclului conține mai mult de o instrucțiune, ea va fi cuprinsă între acoladă deschisă și închisă;

- ✓ Deoarece testarea are loc la sfârșit, corpul unui ciclu cu test final se va executa cel puțin o dată.

Exemplu:

- Să se calculeze suma a n numere naturale citite pe rând de la tastatură (unde n este cunoscut).

Problema a fost deja rezolvată în varianta cu ciclu cu test inițial, iar pseudocodul în varianta cu ciclu cu test final va este de asemenea deja cunoscut din capitolul "Algoritmi. Principiile programării structurate", lecția "Structura repetitivă cu test final". De aceea, considerăm inutile orice alte explicații.

```
#include <iostream.h>
void main ()
{
    int S,i,x,n;
    cout << "\n n="; cin >> n;
    S=0;
    i=1;
    do
    {
        cout << "\n x="; cin >> x;
        S+=x;
        i++;
    } while (i<=n);
    cout << "S=" << S;
```

```
citește n;
S←0;
i←1;
execută
    început
        citește x;
        S←S+x;
        i←i+1;
    sfârșit
cât timp (i<=n);
scrie S;
```



Aplicație R.IV.16. Afișarea unui șir de litere

(Bacalaureat iulie 2001, varianta 1)

Care trebuie să fie valoarea inițială a variabilei întregi i pentru ca următoarea secvență să afișeze șirul "AAAAAA"?

```
do
{
    cout << "AA";
    i--;
} while (i!=3);
```

- a) 0 b) 1 c) 2 d) 3 e) 4 f) 5 g) 6 h) nu există nici o valoare



Rezolvare

La un pas al ciclului do-while se afișează două caractere 'A' {cout << "AA";}, iar în total trebuie tipărite șase astfel de caractere. În consecință, ciclul trebuie să aibă trei pași. Pornind de la o anumită valoare inițială pe care se cere să o aflăm, contorul i se decrementează cu 1 la fiecare pas {i--, adică i=i-1}, până când ajunge la valoarea 3. Observați că imediat după ce i a ajuns la valoarea 3, urmează testarea "i!=3" din linia while, deci afișarea șirului 'AA' nu se face și pentru valoarea 3 (imediat după ce i a primit valoarea 3 prin execuția decrementării i--, urmează testarea condiției din linia

while care înseamnă "3≠3", cu rezultat *FALS*). Drept urmare, valorile lui *i* pentru care are loc afișarea respectivă vor fi cele mai mari decât 3. Întrucât am stabilit că sunt necesare trei astfel de valori, rezultă că acestea vor fi 6, 5, 4. Așadar valoarea inițială a lui *i* va fi 6.

Răspuns corect: g)

Aplicație R. IV.17. Produs de numere impare

Scrieți un algoritm care calculează produsul numerelor naturale pare nenule mai mici sau egale decât o valoare *n* dată.

Exemplu: pentru *n*=8 rezultă *P*=1·3·5·7=105, pentru *n*=9 se va calcula *P*=1·3·5·7·9=945.

Precizare: Această problemă a fost rezolvată și în varianta cu ciclu cu test inițial, în cadrul aplicației R. IV.14. din lecția "IV.4.1. Structura repetitivă cu test inițial".

Rezolvare

Algoritmul a fost explicat în aplicația R. II.23. din lecția "II.4.2. Structura repetitivă cu test final" (capitolul II).

```
#include <iostream.h>

void main ()
{
    int i,n,P;
    cout << "\n n="; cin >> n;
    P=1; i=1;
    do
    {
        P*=i;
        i+=2;
    }
    while (i<=n);
    cout << "\n P=" << P;
}
```

```
citește n;
P←1;
i←1;
execută
    început
        P←P*i;
        i←i+2;
    sfârșit
cât timp (i<=n)
    scrie P;
```

Încercați singuri!

Scrieți programe pentru algoritmii propuși în tema T.10. din lecția "2.4.1. Structura repetitivă cu test inițial" (capitolul II).

Aplicație R. IV.18. Șir de numere a cărui citire se termină cu 0

Se dă un șir de numere citite pe rând de la tastatură, până la întâlnirea valorii 0 (care nu face parte din șir). Presupunând că șirul conține cel puțin un număr nenul, să se afișeze câte numere pozitive sunt în șir.

Precizare: Această problemă a fost rezolvată și în varianta cu ciclu cu test inițial, în cadrul aplicației R. IV.11. din lecția "IV.4.1. Structura repetitivă cu test

inițial", cu deosebirea că acolo numerele se citeau pe rând cât timp nu s'a introdus valoarea 0.

Rezolvare

Algoritmul a fost explicat în aplicația R. II.10. din lecția "II.4.2. Structura repetitivă cu test final" (capitolul II).

```
citește x;
nr←0;
execută
    început
        dacă (x>0) atunci
            nr←nr+1;
        citește x;
    sfârșit
cât timp (x≠0);
scrie nr;
```

Programul:

```
#include <iostream.h>

void main ()
{
    int x,nr;
    cout << "\n x="; cin >> x;
    nr=0;
    do
    {
        if (x>0)
            nr++;
        cout << "\n x="; cin >> x;
    } while (x);
    cout << "\n nr=" << nr;
}
```

Încercați singuri!

Scrieți programe pentru algoritmii propuși în temele T.11. și T.12. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Aplicație R. IV.19. Suma cifrelor unui număr natural

Scrieți un algoritm care determină suma cifrelor unui număr natural *x* citit de la tastatură.

Precizare: Această problemă a fost rezolvată și în varianta cu ciclu cu test inițial, în cadrul aplicației R. IV.13. din lecția "IV.4.1. Structura repetitivă cu test inițial".

Rezolvare

Algoritmul a fost explicat în aplicația R.II.11. din lecția "II.4.2. Structura repetitivă cu test final" (capitolul II).

```

citește x;
d ← x;
S ← 0;
execută
    început
        S ← S + d % 10;
        d ← d / 10;
    sfârșit
cât timp d ≠ 0;
scrie S;
    
```

Programul:

```

#include <iostream.h>

void main()
{
    int S; long d, x;
    // citește de la tastatură numărul x
    cout << "\n x="; cin >> x;
    // inițial deimpartitul d este chiar numărul x, iar suma cifrelor S este 0
    d = x; S = 0;
    // extrage cifrele lui x într-un ciclu, adăugând la S fiecare cifră
    do
    {
        S += d % 10; // adăuga cifra extrasă d % 10 la suma cifrelor S
        d = d / 10; // catul împărțirii curente, d / 10, va fi deimpartit pentru împărțirea următoare
    } while (d); // cât timp deimpartitul este diferit de 0
    cout << "\n S=" << S;
}
    
```

Încercați singuri!

- 1. Scrieți programe pentru algoritmi propuși în tema T.13. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).
- 2. Scrieți programe pentru algoritmi propuși în tema T.14. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Aplicație R.IV.20. Multiplii comuni

(Bacalaureat iulie 2008, varianta 34)

Se citesc de la tastatură trei numere naturale nenule x , y și p , având cel mult trei cifre fiecare. Scrieți un algoritm care afișează pe ecran primii p multiplii comuni ai celor două numere x și y .

Exemplu: pentru $x=180$, $y=120$ și $p=5$, se vor afișa numerele: 360, 720, 1080, 1440, 1800.

Rezolvare

Pseudocodul acestui algoritm a fost explicat pe larg în cadrul aplicației R.II.27. din cadrul lecției "II.4.2. Structura repetitivă cu test final" (capitolul II).

Pseudocodul:

```

citește x, y, p;
dacă (x >= y) atunci
    început
        a ← x; b ← y;
    sfârșit
    altfel
        început
            a ← y; b ← x;
        sfârșit
    execută
        r ← a % b;
        a ← b;
        b ← r;
    cât timp (r ≠ 0);
d ← a;
m ← (x * y) / d;
i ← 1;
execută
    scrie i * m, ' ';
    i ← i + 1;
cât timp (i <= p);
    
```

Programul:

```

#include <iostream.h>
#include <conio.h>
void main ()
{
    unsigned int x, y, p, a, b, d, r, m, i;
    cout << "Dati numerele x, y si p: " << endl;
    cin >> x >> y >> p;
    // aflăm d = c.m.m.d.c. al lui x și y, cu algoritmul lui Euclid
    if (x >= y)
    {
        a = x; b = y;
    }
    else
    {
        a = y; b = x;
    }
    do
    {
        r = a % b;
    }
    
```

```

a=b;
b=r;
} while (r);
d=a;
// aflam m = c.m.m.m.c. al lui x si y
m=(x*y)/d;
// aflam primii p multiplii comuni ai lui x si y, care vor fi m, 2m, 3m, ..., pm
cout << endl << "Primii " << p << " multiplii sunt:" << endl;
i=1;
do
{
    cout << i*m << " ";
    i++;
} while (i<=p);
getch();
}

```

Încercați singuri!

Scrieți programe pentru algoritmi propuși în tema T.27. din lecția "II.4.2. Structura repetitivă cu test final" (capitolul II).

Ați înțeles?

Probleme cu răspuns scurt

Următoarele două probleme se referă la secvența de program de mai jos în care toate variabilele memorează numere naturale nenule.

```

cin >> n;
x=1;
y=x-1;
do
{
    y=x*(x-1)+y; x++;
} while (x<=n);

```

► 1. (Bacalaureat iulie 2008, varianta 3)

De câte ori este evaluată expresia logică din linia "until" în timpul execuției secvenței?

- a) de n^2 ori b) de $n+1$ ori c) o dată d) de n ori

► 2. Dacă de la tastatură se citește $n=5$, care vor fi valorile finale ale variabilelor x și y , după execuția secvenței?

► 3. (Bacalaureat iulie 2008, varianta 4)

Care vor fi valorile finale ale variabilelor a și b afișate la finele execuției secvenței de program următoare?

```

a=1; b=0;
do
{
    b++;
    a*=b;
} while (a<125);

```

Următoarele două probleme se referă la secvența de program de mai jos în care toate variabilele memorează numere naturale nenule.

```

cin >> a;
b=0;
do
{
    a--; b+=a*a;
} while (a!=0 && b<=99);
cout >> a >> b;

```

► 4. (Bacalaureat iulie 2008, varianta 15)

Dacă de la tastatură se citește numărul $a=5$, ce valori se vor afișa în urma execuției secvenței?

► 5. Găsiți un număr cu exact două cifre care ar putea constitui valoarea citită în variabila a , astfel încât valoarea finală a lui b să reprezinte suma pătratelor primelor a numere naturale $(1^2+2^2+\dots+a^2)$.

► 6. (Bacalaureat iulie 2006, varianta 2)

Ce reprezintă rezultatul afișat în urma execuției secvenței de program de mai jos?

```

cin << a;
do
{
    c=floor(a); a=(a-c)*10;
} while (floor(a)>0);
cout >> a;

```

- a) prima zecimală nenulă a lui a b) prima zecimală a lui a
c) ultima zecimală nenulă a lui a d) ultima zecimală a lui a

Următoarele patru probleme se referă la secvența de program de mai jos în care toate variabilele memorează numere naturale nenule.

```

cin >> n;
m=0;
v=n;
u=n%10
do
{
    c=n%10;
    v=v*10+c;
    if (c==u)
        m++;
    n=n/10;
} while (n);
cout >> v >> m;

```

► 7. (Bacalaureat iulie 2009, varianta 65)

Deduceți ce valori se vor afișa pe ecran, în cazul în care de la tastatură se citește numărul $n=1327$.

► 8. (Bacalaureat iulie 2009, varianta 65)

Găsiți o valoare care ar putea fi citită în variabila n , astfel încât valoarea afișată pentru variabila m să fie 3.

► 9. Știind că valorile variabilelor folosite în cadrul secvenței pot fi orice numere naturale cu maxim trei cifre, este corectă instrucțiunea de mai jos prin care se declară respectivele variabile? Justificați răspunsul.

```
int m,n,u,v,c;
```

► 10. Dacă de la tastatură se citește valoarea $n=688$, câte atribuiri se execută în total în timpul rulării secvenței?

► 11. Dați un exemplu de număr care ar putea fi citit în variabila n , astfel încât valoarea finală a variabilei m să fie 0.

► 12. (Bacalaureat iulie 2001, varianta 5)

Care trebuie să fie valoarea inițială a variabilei întregi i pentru ca secvența alăturată să afișeze șirul XXX?

- a) 0 b) 1 c) 2 d) 3 e) 4
f) 5 g) 6 h) nu există nici o valoare

► 13. (Bacalaureat iulie 2000, variantele 4 și 9)

Care este valoarea variabilei întregi a pentru care secvența de program alăturată va afișa exact un caracter 'A'?

- a) 12 b) 5 c) 4 d) 1

► 14. Scrieți o secvență de program care repetă citirea a unei variabile întregi a până când introducem de la tastatură o valoare pară și divizibilă cu 5.

```
a) do
    cin >> a;
    while (a % 2 == 0 || a % 5 == 0);
```

```
c) do
    cin >> a;
    while (a % 2 != 0 || a % 5 != 0);
```

```
b) do
    cin >> a;
    while (a % 2 != 0 && a % 5 != 0);
```

```
d) do
    cin >> a;
    while (a % 10 == 0);
```

► 15. De câte ori afișează valoarea 0 programul următor?

```
int i=2;
do
{
    i+=2;
    cout << "\n 0";
} while (i<=5);
i--;
```

- a) nici o dată
b) o dată
c) de două ori
d) de trei ori

► 16. Ce valoare va afișa programul următor?

```
int x=5;
do
{
    if (x % 3 == 0)
        x+=2;
    else x--;
    cout << '1';
} while (x<10);
```

- a) nimic
b) 1
c) 11
d) programul va intra într-un ciclu infinit

► 17. Programul de mai jos va tipări numărul

```
int n=10;
while (n>0)
do {
    n--;
} while (n!=-2);
cout << n;
```

- a) 10
b) 2
c) -2
d) -3
e) programul conține un ciclu infinit

► 18. Care dintre secvențele de mai jos afișează corect numerele divizibile cu 7 situate în intervalul închis $[a,b]$?

```
a)
i=a;
do
{
    i++;
} while (i % 7 != 0);
while (i<=b)
{
    cout << i << ' ';
    i=i / 7;
    cout << i << ' ';
    i+=7;
}
```

```
b)
i=b;
while (i % 7 != 0)
    i--;
do
{
    cout << i << ' ';
    i=i / 7;
} while (i>=a);
```

```
c)
i=a;
while (i<b)
    if (i % 7 == 0)
        cout << i << ' ';
    i++;
```

d) Nici una dintre variantele anterioare.



Probleme propuse

Folosind cicluri cu test final, scrieți programe pentru toți algoritmi propuși în tema T.R.3. de la sfârșitul capitolului II (secțiunea "Probleme recapitulative pentru toate tipurile de cicluri").



IV.2.3. Ciclu cu număr fix de pași (for)

Într-un astfel de ciclu, o secvență de instrucțiuni se execută de un anumit număr de ori bine determinat, stabilit cu ajutorul valorilor pe care le parcurge un așa-numit *contor*.

În lecția "Structura repetitivă cu număr fix de pași" din capitolul "Principiile programării structurate", am prezentat pentru acest ciclu o sintaxă foarte simplă, neconformă cu cea impusă de limbajul C++.

pentru (<contor> ← <v<sub>12nexecută
<secv></sub>

Am prezentat atunci **funcționarea de principiu**: *contorul ciclului* <contor> parcurge pe rând valorile <v_{12ncorpul ciclului (adică se execută un pas al ciclului).}

În C++, linia "pentru" este alcătuită din trei expresii. Sintaxa unui ciclu cu contor arată astfel:

```
for (<expr_1>; <expr_2>; <expr_3>)
    <secv>
```

- ✓ <contor> este o variabilă numită *contorul ciclului*;
- ✓ <secv> este o secvență formată dintr-una sau mai multe instrucțiuni, numită *corpul ciclului*;
- ✓ <expr_1> este așa-numita "expresie de inițializare a ciclului", din care trebuie să rezulte valoarea inițială a contorului (valoarea cu care va intra în ciclu și va executa primul pas);
- ✓ <expr_2> reprezintă condiția de continuare a ciclului, mai exact condiția pe care trebuie să o îndeplinească contorul pentru ca execuția ciclului să continue cu un nou pas; prin această expresie ne putem asigura că ciclul este unul finit: el se va încheia în momentul în care condiția dată de <expr_2> nu mai este adevărată;
- ✓ <expr_3> asigură trecerea la pasul următor al ciclului; mai exact, printr-o incrementare sau atribuire, această expresie stabilește valoarea pe care o va avea contorul la pasul următor al ciclului.

Exemplu:

Să vedem cum să transcriem în C++ un ciclu cu contor care în pseudocod este descris astfel.

```
pentru (i ← 1, 2, ..., n)          for (<expr_1>; <expr_2>; <expr_3>)
    <secv>                          <secv>
```

Contorul *i* al ciclului parcurge pe rând valorile 1, 2, ..., *n*-1, *n*, și, la fiecare pas, pentru fiecare valoare a lui *i*, se execută corpul ciclului reprezentat de secvența de instrucțiuni <secv> (atenție, valoarea variabilei *n* trebuie să fie cunoscută înainte de "intrarea" în ciclu). Tot ce trebuie să facem este să stabilim care sunt expresiile <expr_1>, <expr_2> și <expr_3> care apar în transcrierea C++ a liniei "pentru".

• Expresia <expr_1> trebuie să stabilească prima valoare a contorului *i*, cu care se va executa primul pas al ciclului. Se vede clar că această valoare este 1, deci <expr_1> este simpla atribuire *i*=1.

• Expresia <expr_2> trebuie să ne dea condiția de continuare a ciclului. Contorul *i* pleacă de la valoarea 1, crește la fiecare pas cu câte o unitate, până în momentul în care a ajuns la valoarea *n* inclusiv. Altfel spus, ciclul se oprește atunci când valoarea lui *i* a

depășit *n*. În concluzie, ciclul se reia atâta timp cât valoarea lui *i* este mai mică sau egală decât *n*, deci condiția de continuare a ciclului este "*i* ≤ *n*".

• Expresia <expr_3> are rolul de modifica contorul în așa fel încât să se asigure trecerea la pasul următor al ciclului; Observați că la fiecare nou pas, valoarea lui *i* crește cu o unitate față de pasul precedent, deci trecerea la un nou pas se face prin incrementarea contorului *i* cu 1. Prin urmare, această expresie este "*i*++".

Acum putem traduce în C++ ciclul nostru:

```
for (i=1; i<=n; i++)
    <secv>
```

Evident că în loc de "*i* ≤ *n*", puteam scrie o condiție echivalentă, cum ar fi "*i* < *n*+1"!

Observații:

♦ În cazul în care corpul ciclului are cel puțin două instrucțiuni, acesta trebuie cuprins între acolade.

♦ Expresiile <expr_1>, <expr_2> și <expr_3> pot fi oricât de complexe; ele pot conține și alte operații în afara celor care stabilesc rolul lor; astfel de situații mai speciale, care constituie "farmecul" C-ului, vor face obiectul secțiunii "Extindere" de la sfârșitul lecției.

♦ Nu se va altera contorul unui ciclu **for**, în corpul acestuia (prin atribuire sau citire de la tastatură).

Exemplu:

Reluăm aceeași problemă rezolvată până acum cu celelalte două tipuri de cicluri:

Să se calculeze suma a *n* numere naturale citite pe rând de la tastatură (unde *n* este cunoscut).

Notând contorul ciclului cu *i*, acesta va lua ca valori toate numerele naturale de la 1 la *n* (*i*=1, 2, ..., *n*). "Traducerea" în C++ a liniei de pseudocod "pentru (*i* ← 1, 2, ..., *n*)" a fost explicată deja: "for (*i*=1; *i* ≤ *n*; *i*++)". Pentru fiecare dintre aceste valori ale lui *i* se execută corpul ciclului, care constă în citirea unui număr *x* și adăugarea lui la suma *s* (unde *s* s-a inițializat cu 0 înainte de ciclu). Observăm că variabila-contor *i* nu are decât rolul de a număra execuțiile corpului ciclului.

```
#include <iostream.h>

void main ()
{
    int i, n, x, s;
    cout << "\n n=";
    cin >> n;
    s=0;
    for (i=1; i<=n; i++)
    {
        cout << "\n x= ";
        cin >> x;
        s+=x;
    }
}
```

```
citește n;
s ← 0;
pentru (i ← 1, 2, ..., n) execută
    început
    citește x;
    s ← s + x;
    sfârșit
scrie s;
```

```

}
cout << "\n S= " << S;
}

```

Exemplu de funcționare:

Să urmărim funcționarea ciclului pas cu pas, dacă $n=3$ iar numerele citite succesiv în variabila x sunt 2, 5, 8.

Înainte de ciclul avem $S=0$.

La "intrarea" în ciclu se execută mai întâi expresia care inițializează contorul: $i=1$. Astfel începe primul pas al ciclului. Mai departe se testează condiția de continuare " $i \leq n$ ": " $1 \leq 3$? da" $\Rightarrow$ continuă primul pas, cu corpul ciclului: citește $x=2$ și $S=S+x=0+2=2$. Urmează modificarea contorului pentru trecerea la pasul următor, prin incrementarea $i++ \Rightarrow i=1+1=2$. Pașii următori se derulează într-o manieră absolut similară:

Pasul 2: $i \leq n$? $2 \leq 3$? da $\Rightarrow$ citește $x=5$ și $S=S+x=2+5=7$; $i++ \Rightarrow i=2+1=3$

Pasul 3: $i \leq n$? $3 \leq 3$? da $\Rightarrow$ citește $x=8$ și $S=S+x=7+8=15$; $i++ \Rightarrow i=3+1=4$

Pasul 4: $i \leq n$? $4 \leq 3$? nu $\Rightarrow$ se încheie ciclul.

Observație:

Dintre artificii pe care le permite limbajul C++, unul este foarte des folosit de către programatori. Este vorba despre faptul că în expresia `<expr_1>` de inițializare a contorului pot fi incluse și alte inițializări. În cazul nostru, atribuirii $i=1$ care inițializează contorul, îi putem atașa și inițializarea lui S cu 0 (făcută înainte de ciclul). Astfel, în loc de

```

S=0;
for (i=1; i<=n; i++)
{
    cout << "\n x= ";
    cin >> x;
    S+=x;
}

```

putem scrie:

```

for (S=0, i=1; i<=n; i++)
{
    cout << "\n x= ";
    cin >> x;
    S+=x;
}

```

Observați că expresia `<expr_1>` de inițializare a contorului a devenit " $S=0, i=1$ ". Veți putea înțelege cu ușurință această expresie dacă ați parcurs lecția "Operatorul <virgulă>" din secțiunea "Extindere" a capitolului II, "Elemente de bază ale limbajului C++". Pe scurt, două expresii elementare, " $S=0$ " respectiv " $i=1$ ", au fost "legate" între ele prin operatorul "virgulă". Asta înseamnă că ele se vor executa succesiv, în ordinea în care au fost scrise, adică mai întâi $S=0$ apoi $i=1$. Deosebirea față de varianta precedentă constă în faptul că, în loc să se execute înainte de ciclul, atribuirea $S=0$ are loc la începutul primului pas, dar acest fapt nu influențează evoluția ulterioară a lucrurilor.

Alte exemple de cicluri for:

```

1) char c;
   for (c='A'; c<='D'; c++)
       cout << c;

```

• Contorul ciclului este variabila c de tipul caracter. Prima valoare a sa este caracterul 'A', ciclul se execută atâta timp cât caracterul memorat în variabila c este mai mic sau egal decât 'D', și la fiecare pas se incrementează contorul c . În condițiile în care inițial $c='A'$, după prima incrementare valoarea contorului c va fi caracterul 'B'. De ce? Reamintim că în setul ASCII, fiecare caracter are un cod numeric, numit codul ASCII al caracterului, iar limbajul C++ face o asociere între caracter și codul său. Astfel, atunci când trebuie să incrementeze caracterul, el va incrementa de fapt codul său ASCII. În cazul nostru, codul lui 'A' pe care îl incrementează este 65, în urma incrementării ajunge la valoarea 66 care este codul lui 'B', deci, în virtutea asocierii inverse cod $\rightarrow$ caracter, noua valoare a variabilei c va fi caracterul 'B'. În acest mod, contorul c va parcurge succesiv caracterele 'A', 'B', 'C', 'D'. Odată ajuns la 'D', ciclul se oprește datorită condiției de continuare $c \leq 'D'$.

```

2) S=0;
   int n=9, S=0, i;
   for (i=n; i>=1; i-=2)
       S+=i;

```

• Prima valoare a lui i este n adică 9, la fiecare pas contorul i scade cu două unități prin atribuirea $i=i-2$, iar ciclul se execută atâta timp cât $i \geq 1$. Prin urmare, valorile lui i vor fi succesiv 9, 7, 5, 3, 1. La fiecare pas, se adaugă valoarea curentă a lui i la o variabilă S (inițializată cu 0). V-ați dat seama desigur că, în final, în variabila S se va calcula suma $S=9+7+5+3+1=25$. Am ales acest exemplu pentru a ilustra faptul că într-un ciclu "for", contorul se poate modifica de la un pas la altul, după orice regulă, nu neapărat prin incrementare cu o unitate.

```

3) int i, n=4;
   for (i=sqrt(3); i<=n; i++)
       cout << "\n " << i;

```

• Conform primei expresii din linia "for", prima valoare a contorului i ar trebui să fie "radical din 3", adică 1.732... Această valoare va fi însă trunchiată la 1, urmare a faptului că variabila i căreia i se atribuie este de tipul `int` (regula conversiilor implicite). La fiecare pas se incrementează i cu 1, iar condiția de continuare este $i \leq n$. Rezultă că valorile lui i vor fi succesiv 1, 2, 3, 4 care se afișează în ciclu.

Oricât ar părea de ciudat, acest ciclu funcționează și în cazul în care contorul i se declară de tipul `float`. În această situație, prima valoare a lui i nu mai este trunchiată, afișându-se pe rând 1.732051, 2.732051, 3.732051, 4.732051 (instrucțiunea `cout` tipărește implicit primele șase zecimale).

Aplicație R.IV.21. Secvențe

Care dintre secvențele de program de mai jos nu afișează 123 ?

```
a) n=3;
   for (i=1; i>=n; i++)
       cout << i;

c) for (a='1'; a<='3'; a++)
       cout << a;
```

```
b) n=7;
   for (i=1; i<=n/2; i++)
       cout << i;

d) F=1; E=0;
   for (i=3; i>=1; i--)
   {
       E=E+i*F; F=F*10;
   }
   cout << E;
```

Rezolvare

Variantă a) este eronată: ciclul "for" nu are nici un pas, deoarece condiția de continuare "i>=n" este falsă încă de la început, pentru prima valoare a contorului i=1. Toate celelalte variante afișează 123 sub o formă sau alta (atenție, nu se cere să se formeze neapărat numărul întreg 123 și să se memoreze într-o variabilă!).

Condiția de continuare a ciclului din varianta b) este "i<=3" (rezultatul împărțirii 7/2 va fi 3, deoarece atunci când împarte doi întregi compilatorul va furniza câtul întreg al împărțirii). Prin urmare, valorile contorului i vor fi pe rând 1, 2, 3, și aceste valori se tipăresc "lipite" una de alta, simulându-se astfel numărul 123.

Contorul a al ciclului din varianta c) parcurge pe rând caracterele '1', '2', '3' și la fiecare pas se afișează valoarea contorului. Efectul final va fi tipărirea caracterelor '1', '2' și '3', "lipite" unul de altul.

În varianta d) se formează numărul întreg 123 în variabila E, care apoi se afișează. Să parcurgem pașii ciclului din această secvență, în care i=3, 2, 1 și inițial F=1, E=0:

- ✓ i=3: E=E+i*F=0+3*1=3, F=F*10=1*10=10
- ✓ i=2: E=E+i*F=3+2*10=23, F=F*10=10*10=100
- ✓ i=1: E=E+i*F=23+1*100=123, F=F*10=100*10=1000

Răspuns corect: a).

Aplicație R.IV.22. Număr prim

Scrieți un algoritm care verifică dacă un număr natural x diferit de 0 și 1, este prim sau nu, afișând un mesaj corespunzător.

Precizare: Această problemă a fost rezolvată și în varianta cu ciclu cu test inițial, în cadrul aplicației R.IV.15. din lecția "IV.4.1. Structura repetitivă cu test inițial".

Rezolvare

Algoritmul a fost explicat în aplicația R.II.31. din lecția "II.4.1. Structura repetitivă cu număr fix de pași" (capitolul II).

```
citește x;
ok←1;
pentru (i←2,...,x/2) execută
    dacă (x%i==0) atunci
        ok←0;
```

```
dacă (ok==1)
    scrie 'Numărul este prim'
altfel
    scrie 'Numărul NU este prim';
```

Programul:

```
#include <iostream.h>

void main ()
{
    int x,i,ok;
    cout << "\n x=";
    cin >> x;
    ok=1;
    for (i=2; i<=x/2; i++)
        if (x % i==0)
            ok=0;
    if (ok)
        cout << "\n Numarul este prim";
    else
        cout << "\n Numarul NU este prim";
}
```

Încercați singuri!

Scrieți un program pentru algoritmi propuși în tema T.18. din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Alți algoritmi care folosesc structuri repetitive cu contor

Aplicație R.IV.23. Numere pare și divizibile cu 5

(Bacalaureat iulie 2008, varianta 27, enunț adaptat)

Scrieți un program care, pentru un număr natural n nenul de cel mult patru cifre citit de la tastatură, afișează, în ordine crescătoare primele n numere naturale pare și divizibile cu 5, precum, și suma acestora.

Exemplu: pentru n=6 se vor afișa numerele 10 20 30 40 50 60

Rezolvare

Pseudocodul acestui algoritm a fost explicat pe larg în cadrul aplicației R.II.33. din cadrul lecției "II.4.3. Structura repetitivă cu număr fix de pași" (capitolul II).

Pseudocodul:

```
citește n;
S←0;
```

```

pentru (k←1,2,...,n) execută
    început
        scrie 10*k;
        S←S+10*k;
    sfârșit
scrie S;

```

Programul:

```

// Bac 2008, III.1 / var 27
#include <iostream.h>
#include <conio.h>

void main ()
{
    clrscr();
    int k,n,S=0; // S = suma numerelor afisate
    // citeste n
    cout << "Dati valoarea lui n: ";
    cin >> n;
    cout << "Numerele cerute sunt: " << endl;
    // contorizeaza, afiseaza si insumeaza primele n numere pare si divizibile cu 5
    // adica divizibile cu 10
    for (k=1; k<=n; k++)
    {
        cout << (10*k) << " ";
        S+=10*k;
    }
    cout << endl << "Suma numerelor este " << S;
    getch();
}

```

Încercați singuri!

Folosind structuri repetitive cu contor, scrieți programe pentru toate problemele propuse în cadrul temei T.20., din lecția "II.4.1. Structura repetitivă cu test inițial" (capitolul II).

Aplicație R.IV.24. Șirul lui Fibonacci

Să se afișeze primii n termeni ai șirului lui Fibonacci. Șirul are primii doi termeni egali cu 1 și fiecare din următorii termeni este egal cu suma dintre termenul precedent și termenul ante-precedent.

Rezolvare

Algoritmul a fost explicat în aplicația R.II.34. din lecția "II.4.3. Structura repetitivă cu număr fix de pași" (capitolul II).

```

citește n;
prec←1;
anteprec←1;

```

```

scrie prec,anteprec;
pentru i←3,...,n
    început
        fib←prec+anteprec;
        scrie fib;
        anteprec←prec;
        prec←fib;
    sfârșit

```

Programul:

```

#include <iostream.h>

void main ()
{
    int i,n;
    long fib,prec,anteprec;
    // n=numarul de termeni
    // fib=termenul, prec=precedentul, anteprec=anteprecedentul
    cout << "\n n="; cin >> n;
    prec=anteprec=1;
    // primii doi termeni
    cout << "\n " << anteprec << " " << prec;
    for (i=3; i<=n; i++)
    {
        // restul termenilor
        fib=prec+anteprec;
        cout << "\n " << fib;
        anteprec=prec;
        prec=fib;
    }
}

```

Încercați singuri!

Scrieți programe pentru algoritmi propuși în tema T.34. din lecția "II.4.3. Structura repetitivă cu număr fix de pași" (capitolul II).

Aplicație R.IV.25. Triunghi de numere

Să se afișeze triunghiul de numere alăturat, pentru o valoare dată a lui n .

```

1
1 3
1 3 5
.....
1 3 5 ..... 2n-1

```

Rezolvare

Algoritmul problemei a fost explicat pe larg în pseudocod în cadrul problemei rezolvate R.II.35. din secțiunea "Aprofundare", capitolul "II.4.3. Structura repetitivă cu număr finit de pași".

Pseudocodul:

```

citește n;
pentru (i←1,...,n)
    început
        pentru (j←1,...,i)
            scrie 2*j-1:2, ' ';
        <salt la rând nou>;
    sfârșit;
sfârșit
    
```

Programul:

```

#include <iostream.h>

void main ()
{
    int i,j,n;
    cout << "\n n=";
    cin >> n; // n este numarul de randuri
    for (i=1;i<=n; i++) // parcurge liniile i
    {
        for (j=1; j<=i; j++) // afiseaza valorile ficcarei linii i
            cout << (2*j-1) << ' ';
        cout << "\n";
    }
}
    
```

Încercați singuri !

Scrieți programe pentru algoritmi propuși în tema T.35. din lecția "II.4.3. Structura repetitivă cu număr fix de pași" (capitolul II).

Ați înțeles ? Probleme cu răspuns scurt

► 1. Fie următoarele declarații de variabile:

```

int i,a=5, d=10;
b:char;
    
```

Să se precizeze care din următoarele instrucțiuni for sunt corecte:

- a) for (i=1; i<=a; i++) cout << 1;
- b) for (b='m'; b<='v'; b--) cout << 1;
- c) for (i=a/2; i<=floor(d); i=2) cout << 1;
- d) for (d==0; d<=7; d++) cout << 1;

► 2. Câte valori distincte va afișa instrucțiunea:

```

for (i=1; i<=22; i++) cout << "\n" << floor(sqrt(i));
    
```

- a) 22
- b) 21
- c) 5
- d) 4

► 3. (Bacalaureat iulie 2000, varianta 7)

Care va fi valoarea afișată de către programul următor dacă a=b=3 ?

```

#include <iostream.h>
void main()
{
    long a,b,c,z,i;
    cout << "Dati a si b: ";
    cin >> a >> b;
    c=0; z=1;
    for(i=1;i<=a;i++)
    {
        c+=z; z*=b;
    }
    cout << "c=" << c;
}
    
```

- a) 8
- b) 32
- c) 27
- d) 13

► 4. Câte numere afișează programul următor ?

```

int i,j,nr=0;
for (i=1; i<=5; i++)
    for (j=5; j>=1; j--)
    {
        nr+=1;
        if (i==j-i) cout << "\n" << nr;
    }
    
```

- a) două
- b) trei
- c) patru
- d) cinci

► 5. (Bacalaureat august 2000, varianta 1)

Care dintre următoarele secvențe de instrucțiuni atribuie variabilei întregi x valoarea 10^n , cu n număr natural, variabila auxiliară i fiind de tip întreg ?

- a) x=1; i=1; do { x*=10; i++; } while (i<n);
- b) x=1; for (i=n; i>0; i--) x*=10;
- c) x=10; for (i=1; i<=n; i++) x*=i;
- d) x=1; i=0; while (i<=n) { i++; x*=i; }

► 6. (Bacalaureat august 2000, varianta 3)

Știind că x și i reprezintă două variabile întregi, stabiliți care dintre următoarele secvențe de instrucțiuni afișează numărul 11.

- a) x=7; for (i=1; i<=5; i++) x++; cout << x;
- b) x=1; while (x<10) x++; cout << x;
- c) x=1; for (i=1; i<=2; i++) cout << x; cout << "\n";
- d) x=1; i=2; while (i>=0) { cout << x; i--; }

► 7. (Bacalaureat iulie 2001, varianta 4)

Știind că x și y reprezintă două variabile întregi și că x are valoarea 11, precizați care dintre următoarele secvențe afișează exact zece caractere 'A'.



Capitolul V: Tablouri



V.1. Vectori (tablouri unidimensionale)



V.1.1. Noțiuni introductive

Noțiunea de vector

În practică, apare deseori necesitatea de a prelucra un set de valori de același tip, așezate într-o anumită ordine. O astfel de structură se numește **șir**, iar valorile respective se numesc **elementele șirului**. De exemplu, cu mediile generale ale elevilor unei clase putem alcătui un șir de numere reale.

De la început trebuie observat faptul că ar fi total inefficient să declarăm câte o variabilă pentru fiecare element al șirului, deoarece numărul acestor elemente poate fi foarte mare. De aceea, limbajul Pascal oferă posibilitatea de a memora toate elementele unui șir într-o singură variabilă, în care *elementele sunt dispuse într-o anumită ordine și fiecare element ocupă o anumită poziție bine determinată*. O astfel de variabilă se numește **tablou unidimensional** sau **vector**.

Exemplu: Considerăm șirul de numere întregi -3, 6, -5, 14.

pozițiile (indicii) elementelor
în cadrul vectorului

0	1	2	3
-3	6	-5	14

elementele vectorului

Așa cum am spus, putem memora toate elementele șirului într-o variabilă numită vector, pe care o notăm cu **v**. Elementele șirului vor ocupa poziții succesive în vectorul **v**, începând cu poziția 0 (0, 1, 2, 3, ...).

Pentru a referi un anumit element din vector, trebuie să scriem numele variabilei vector urmat de poziția elementului, cuprinsă între paranteze pătrate. Astfel, pentru vectorul ilustrat mai sus, pozițiile elementelor sunt 0, 1, 2, 3, și:

- Elementul de pe poziția 0 în vector este primul element al șirului, adică -3, și se notează **v[0]**;

- Elementul de pe poziția 1 în vector este al doilea element al șirului, deci **v[1]=6**.
- Analog, **v[2]=-5**, **v[3]=14**.

Pe caz general, dacă variabila vector este **v**, atunci prin **v[i]** înțelegem elementul de pe poziția **i** în vector.

Aici trebuie observat un aspect foarte important. Poziția (indicele) unui element NU arată al câtelea este elementul respectiv în vector, deoarece numerotarea pozițiilor începe de la 0. Astfel, în exemplul anterior, elementul **v[3]**, de pe poziția 3 (sau de indice 3), nu este al treilea element al vectorului, ci al patrulea, înaintea sa aflându-se trei elemente, și anume **v[0]**, **v[1]** și **v[2]**. Majoritatea lucrărilor de specialitate asimilează complet noțiunea de poziție cu cea de indice, chiar dacă acest fapt contravine sensului lingvistic al cuvântului "poziție". Dacă vrem să păstrăm concordanța avem o soluție foarte simplă: nu folosim elementul **v[0]** ! Dacă am utiliza elementele doar începând cu **v[1]**, atunci **v[1]** ar fi primul element, **v[2]** ar fi al doilea, ș.a.m.d., iar poziția fiecărui element ar indica într-adevăr al câtelea este el în vector. În capitolul de față vom adopta această soluție în unele aplicații, acolo unde ne va ajuta să urmărim mai ușor parcurgerea și prelucrarea vectorului.

Poziția unui element se mai numește și **indicele** sau **rangul** elementului, iar elementele se mai numesc și **componente** ale vectorului.

Declararea unui vector

Un vector trebuie declarat, la fel ca orice variabilă, în secțiunea de declarații a programului.

Orice vector se caracterizează printr-un **număr maxim de elemente**. Acesta este o valoare constantă, reprezentând cel mai mare număr de elemente care s-ar putea memora în vectorul respectiv. Acesta trebuie precizat la declararea vectorului, între paranteze pătrate. Pozițiile elementelor vor fi numerotate începând de la 0 până la "numărul maxim de elemente minus 1" (de exemplu, dacă vrem să declarăm un vector cu *maxim* 20 de elemente, atunci pozițiile elementelor vor fi 0, 1, 2, ..., 19).

Pe lângă **numărul maxim de elemente**, în declararea unui vector mai trebuie să apară: **identificatorul vectorului**, (numele variabilei-vector) și **tipul elementelor**.

Exemplu: `int v[20];`

Am declarat un vector **v** cu maxim 20 de elemente numere întregi;

- ❑ identificatorul **v** reprezintă numele variabilei-vector;
- ❑ cuvântul cheie **int** desemnează tipul elementelor vectorului, adică elementele sunt numere întregi.
- ❑ Elementele vectorului sunt **v[0], v[1], ..., v[19]**, având indicii (pozițiile) 0, 1, 2, ..., 19.

Cu aceste precizări, putem prezenta sintaxa generală a declarării unui vector:

Sintaxa: `<tip_elem> <nume>[nr_max_de_elem];`

unde:

- `<tip_elem>` → tipul de date al elementelor;
- `<nume>` → numele (identificatorul) variabilei-vector;
- `<nr_max_de_elem>` → numărul maxim al elementelor vectorului.

Tipul elementelor poate fi orice tip predefinit sau definit de utilizator.

Numărul real (efectiv) de elemente

Așa cum am spus, din declararea unui vector reiese numărul maxim de elemente ale acestuia, adică cel mai mare număr de elemente care s-ar putea memora în vector.

În urma declarării unui vector, compilatorul rezervă pentru elementele sale o zonă fixă de memorie, care să poată memora numărul maxim de elemente. Astfel, dimensiunea zonei de memorie alocată este egală cu numărul maxim de elemente înmulțit cu numărul de octeți necesari pentru un element.

Spre exemplu, pentru vectorul declarat prin

```
int v[25];
```

se alocă în memorie 50 de octeți, deoarece vectorul are maxim 25 de elemente și fiecare element, fiind un număr întreg, ocupă 2 octeți.

Să presupunem că am scris un program în care am declarat vectorul de mai sus, cu maxim 25 de elemente. Este oare necesar ca de fiecare dată când executăm programul să folosim toate cele 25 de elemente? Firește că nu! *La fiecare execuție a programului, ar trebui să putem memora în vector un alt șir, alcătuit din alte elemente.* Pentru aceasta, introducem o altă noțiune: numărul real de elemente într-un vector.

Numărul elementelor efectiv folosite într-un vector se numește **numărul real (efectiv) de elemente** și se declară ca variabilă, deoarece diferă de la o execuție la alta. De obicei, pe acesta îl vom nota cu n .

În plus, la fiecare execuție a programului, elementele pot primi alte valori. De aceea, în general, numărul real de elemente precum și elementele vectorului se citesc de la tastatură.

Exemplu:

Să considerăm un vector v cu maxim 25 de elemente numere întregi.

- Așa cum am arătat deja, vectorul trebuie declarat ca variabilă specificând numărul maxim de elemente:

```
int v[25];
```

• Dacă notăm cu n numărul real (efectiv) de elemente ale vectorului v , atunci:

- n se declară ca variabilă de tip întreg (`int` sau `long`), și se va citi în corpul programului.
- Elementele efectiv utilizate vor fi $v[0], v[1], \dots, v[n-1], v[n-1]$, și ocupă în cadrul vectorului pozițiile $0, 1, \dots, n-2, n-1$. Cu alte cuvinte, dacă de exemplu am declarat un vector cu maxim 25 elemente numere întregi, atunci, pentru el se alocă în memorie 25 de "căsuțe" (locații) de memorie, deci vor fi definite elementele $(v[0], v[1], \dots, v[24])$, dar din el folosim efectiv numai primele n elemente, în speță $(v[0], v[1], \dots, v[n-1])$ ($n \leq 25$).

Poziții	0	1		n-1		23	24
Elemente	$v[1]$	$v[2]$		$v[n-1]$		$v[23]$	$v[24]$

Prin $v[i]$ se înțelege elementul de pe poziția i în vector, unde variabila i parcurge pe rând valorile $1, 2, \dots, n$.

Elementele unui vector se comportă ca niște variabile simple, deci pot fi citite, afișate și folosite în expresii.

Exemplu:

Pentru vectorul v ilustrat grafic mai sus, iată câteva exemple de instrucțiuni care folosesc elemente ale vectorului:

```
v cout << v[2];
```

Afișează elementul $v[2]$, adică elementul de pe poziția 2 în vectorul v .

```
x = (v[3] + v[5]) / 2;
```

Calculează media aritmetică a elementelor $v[3]$ și $v[5]$, pe care o atribuie variabilei x .

```
if (v[0] > 0) cout << "primul element este pozitiv";
```

Testează dacă elementul $v[0]$ este pozitiv, iar în caz afirmativ afișează un mesaj.

Inițializarea unui vector la declarare

Un vector v poate fi inițializat la declarare, adică în momentul declarării sale putem preciza niște valori inițiale ale elementelor.

În C++, pentru a inițializa un vector, este suficient ca la sfârșitul declarației vectorului să scriem caracterul "egal", urmat de enumerarea între două acolade a valorilor inițiale ale elementelor, în ordinea naturală a indicilor, separate între ele prin "virgulă". Evident, respectivele elemente pot fi modificate ulterior în cursul programului, prin atribuirii sau citiri.

Nu este necesar să inițializăm toate elementele! Acele elemente care nu primesc valori la declarare vor fi inițializate automat de către compilator cu valoarea zero.

Iată în continuare un mini-program sugestiv:

Exemplu:

```
#include <iostream.h>
void main()
{
    int v[8]={7,-3,5,-1};
    v[2]++;
    cout << v[1] << " " << v[2] << " " << v[5];
}
```

– Instrucțiunea `{int v[8]={7,-3,5,-1};}` declară un vector `v` cu maxim opt elemente, toate numere întregi, dintre care primele patru sunt inițializate în ordine astfel: `v[0]=7`, `v[1]=-3`, `v[2]=5` și `v[3]=-1`. Restul elementelor vor fi `v[4]=v[5]=v[6]=v[7]=0`, inițializate automat de către compilator.

– Ulterior, în corpul programului, elementul `v[2]` este modificat prin incrementare cu o unitate, deci va rezulta `v[2]=5+1=6`.

– Ultima instrucțiune afișează elementele `v[1]`, `v[2]` și `v[5]` separate printr-un spațiu, adică tipărește valorile -3, 6 și 0.

Observație:

♥ Tipul valorilor inițiale înșiruite între paranteze rotunde la declararea vectorului, trebuie să fie compatibil cu tipul elementelor specificat în cadrul declarației. În caz contrar, compilatorul va face conversiile implicite cunoscute.

Exemplu: `int v[8]={7,-3.92,5,-1};`

Tipul elementelor vectorului `v` este `int`, iar una din valorile inițiale, în speță -3.92 conferită elementului `v[1]`, este un număr real cu parte zecimală. În acest caz, compilatorul pur și simplu va tăia partea zecimală, "forțând" numărul real -3.92 la întregul -3. Așadar în final vom avea `v[1]=-3`.

Aplicație R.V.1 Declarații de vectori

(Bacalaureat iulie 2008, varianta 97)

Dacă `x` este o variabilă ce trebuie să memoreze toți divizorii unui număr natural nenul de cel mult trei cifre, stabiliți care dintre următoarele linii de program reprezintă o declarație corectă a lui `x`.

- a) `int x[3];` b) `long x[50];`
c) `int x;` d) `char x[50];`

Rezolvare

Din capul locului trebuie observat faptul că, fiind vorba despre mai mulți divizori, aceștia vor putea fi memorați doar într-o variabilă indexată de tip vector. În consecință,

varianta c), în care se declară o variabilă simplă `x` de tipul `int`, este eronată. Celelalte trei variante de răspuns reprezintă declararea unui vector `x`, dar:

– în varianta a) se alocă pentru vectorul `x` doar trei "căsuțe", în vreme ce un număr natural cu maxim trei cifre poate avea mai mult de trei divizori;

– în varianta d) se declară un vector `x` cu maxim 50 de elemente, având pozițiile numerotate cu 0, 1, ..., 49, ceea ce este suficient de acoperitor în privința spațiului de memorie necesar (nu există nici un număr de maxim trei cifre care să aibă mai mult de 50 de divizori); în schimb, tipul elementelor acestui vector este `char`, adică elementele ar trebui să fie caractere, în vreme ce divizorii unui număr întreg sunt tot numere!

Prin urmare, răspunsul corect este b). Aici se declară un vector care are tot maxim 50 de elemente, ale căror poziții sunt numerotate începând cu 0, iar tipul elementelor este `long` (divizorii unui număr sunt într-adevăr valori întregi; nu era necesară folosirea tipului "întreg lung", era suficient `int`, dar nici greșit nu este.

Răspuns corect: b)

Ați înțeles? Probleme cu răspuns scurt

► 1. Pentru fiecare dintre afirmațiile de mai jos, stabiliți dacă sunt adevărate sau false:

- a) Pozițiile elementelor pot fi orice numere întregi.
b) Pozițiile elementelor trebuie numerotate obligatoriu începând cu 0, până la o anumită valoare.
c) Din declarația unui vector trebuie să rezulte câte elemente pot fi memorate în el.
d) La declararea unui vector se precizează câte elemente vom folosi efectiv în cadrul programului.
e) La declararea unui vector, la precizarea numărului maxim de elemente putem folosi constante simbolice, dar nu putem folosi variabile.
f) Elementele unui vector pot fi de tipuri diferite.

► 2. Dacă `v` este un vector cu maxim cinci elemente numere întregi, în care pozițiile elementelor sunt 0, 1, ..., 4, scrieți instrucțiunile de atribuire prin care elementele de pe pozițiile pare vor primi ca valori dublul pozițiilor lor.

► 3. Care este deosebirea dintre numărul maxim și numărul real (efectiv) de elemente în cazul unui vector?

► 4. Fie următoarele declarații de variabile:

```
int x;
char v[20], c;
```

Care dintre următoarele atribuiri sunt corecte și care nu? Justificați răspunsul.

- a) `v[2]='m';` b) `v['p']=(x+1)/2;`
c) `v['D']='3';` d) `v[1]=floor(sqrt(x));`
e) `v[3]=c+1;` f) `v[x+1]='m+n';`
g) `x=v[4]+v[5];`

► 5. Se consideră un vector `v` de numere întregi, având pozițiile numerotate succesiv începând cu 0, din care se folosesc efectiv primele `n` elemente, cu `n` număr impar.

5.2.) Pentru $n=41$, elementele vecine cu elementul din mijlocul vectorului sunt:
a) $v[20]$ și $v[21]$ b) $v[20]$ și $v[22]$ c) $v[19]$ și $v[21]$ d) $v[19]$ și $v[20]$

Într-o variabilă x trebuie să memorăm simultan cele 18 medii semestriale ale unui elev la disciplinele studiate, împreună cu media generală, calculată ca medie aritmetică a mediilor semestriale (cu două zecimale exacte). Care este declarația corectă pentru această variabilă?

- a) double a[18];
c) int a[19]
- b) float a[19]
d) long a[20]

Fie vectorii a și b , cu elementele $a=(1,2,4,5,3)$ și $b=(4,1,2,3)$. Atunci $a[b[0]]$ are valoarea:

- a) 5 b) 3 c) 2 d) 1



În cele ce urmează, considerăm vectorul $\mathbf{v}=(v[0], v[1], \dots, v[n-1])$, unde n este numărul real de elemente.

Prin parcurgerea vectorului se înțelege "vizitarea" elementelor pe rând, și prelucrarea acestora. Putem "vizita" toate elementele vectorului sau numai o parte din ele, după cum proiectăm parcurgerea. *Dacă se folosește un ciclu, atunci contorul ciclului va lua pe rând ca valori pozițiile elementelor pe care dorim să le prelucrăm.* În continuare vom nota contorul cu i .

Pentru început, vom ilustra parcurgerea și prelucrarea tuturor elementelor vectorului. Această operație poate fi descrisă astfel:

- Contorul ciclului i va lua pe rând valorile $0, 1, \dots, n-1$, reprezentând pozițiile elementelor în vector;
- La fiecare pas, pentru fiecare valoare a lui i , "vizităm" și prelucrăm elementul $v[i]$.

pentru $(i \leftarrow 0, 1, \dots, n-1)$
 $\langle \text{prelucrează } v[i] \rangle$

Pasul 1: $i=0 \Rightarrow$ prelucrează $v[0]$
Pasul 2: $i=1 \Rightarrow$ prelucrează $v[1]$

Pasul n: $i=n-1 \Rightarrow$ prelucrează $v[n-1]$

```

    pentru (i←1,2,...,n)
        <prelucrează v[i]>

```

Nu putem citi "dintr-o dată" toate elementele vectorului printr-o instrucțiune de genul `{cin >> v};`, pentru că `v` este o variabilă indexată (compusă) care înglobează mai multe valori.

Vom citi mai întâi numărul real de elemente n , apoi citim pe rând elementele $v[0], v[1], \dots, v[n-1]$, folosind ciclul de parcurgere prezentat.

Parcurgem în ciclu pozițiile elementelor $i=0,1,\dots,n-1$ și pentru fiecare valoare a lui i , citim elementul de pe poziția i , adică elementul $v[i]$.

Astfel, pentru $i=0$ se citește elementul $v[0]$, pentru $i=1$ are loc citirea lui $v[1]$, ș.a.m.d.

```
cout << "n=";  
cin >> n;  
for (i=0; i<=n-1; i++)  
{  
    cout << "\n v[" << i << "]=";  
    cin >> v[i];  
}
```

Evident, condiția de trecere la pasul următor " $i \leq n-1$ " din linia **for**, putea fi scrisă și sub forma " $i < n$ " (echivalentă din punct de vedere matematic).

În secvența de program de mai sus, înainte de citirea elementului `v[i]` (cu `cin`), apare linia `{cout << "\n v[" << i << "]=";}`, care afișează un mesaj, cu scopul de a ne spune ce element urmează să citim la pasul respectiv al ciclului. Desigur că o astfel de comandă nu este obligatorie, dar este recomandabilă.

cout << "[\n v [" << i << "]=" ;

↑ ↓ ↓ ↓

salt la rând nou valoarea poziției i

De exemplu, înainte de a aștepta introducerea unei valori pentru elementul `v[2]`, calculatorul va afișa:

 cursor în așteptare

Desigur că se putea afișa un mesaj mult mai simplu, dar mai puțin elegant, de genul:

```
cout << "\n Dati elementul de pe pozitia " << i << ":";
```

Astfel, de exemplu, înainte de citirea elementului $v[2]$, se va afișa textul:

Dati elementul de pe pozitia 2: _

Observație:

Așa cum am precizat în secțiunea introductivă, odată ce am declarat un vector, nu este obligatoriu să folosim toate "căsuțele" acestuia. Astfel, dacă vrem să folosim n elemente din vector, acestea pot fi foarte bine $(v[1], v[2], \dots, v[n])$, în loc de $(v[0], v[1], \dots, v[n-1])$. Singura modificare pe care ar suferi-o algoritmul de citire al vectorului în acest caz, se referă la parcurgerea pozițiilor cu contorul i : acesta va lua valori de la 1 la n . Ciclul de citire arată astfel:

```
for (i=1; i<=n; i++)
{
    cout << "\n v[" << i << "]=";
    cin >> v[i];
}
```

Afișarea unui vector cu n elemente

Nu putem afișa "dintr-o dată" toate elementele vectorului printr-o instrucțiune `{cout << v;}`. Folosim același ciclu de parcurgere a vectorului: vom parcurge pozițiile elementelor din vector $i=0, 1, \dots, n-1$ și pentru fiecare valoare a lui i , afișăm elementul de pe poziția i , adică $v[i]$:

a) Se tipăresc toate elementele pe un singur rând (cu `cout`): la fiecare pas se afișează un element $v[i]$, urmat de un caracter "spațiu".

```
for (i=0; i<=n-1; i++)
    cout << v[i] << ' ';
```

Dacă dorim ca elementele efectiv folosite ale vectorului să fie $(v[1], v[2], \dots, v[n])$, atunci cele două cicluri de mai sus se modifică astfel:

```
for (i=1; i<=n; i++)
    cout << v[i] << ' ';
```

b) Se tipărește fiecare element pe câte un rând: la fiecare pas, înainte de afișarea unui element $v[i]$, cursorul "sare" pe rândul următor.

```
for (i=0; i<=n-1; i++)
    cout << "\n " << v[i];
```

```
for (i=1; i<=n; i++)
    cout << "\n " << v[i];
```

Aplicație R.V.2. Valori distincte nenule

În secvența de program de mai jos, v este un vector cu n elemente numere întregi, care inițial are toate elementele nule, iar i, j, n și k sunt variabile de tip întreg. Câte valori *distincte nenule* va conține vectorul după executarea secvenței date?

```
n=5; k=3;
for (i=0; i<=n-1; i++)
    for (j=0; j<=k-1; j++)
        v[i+j]=k+i;
```

- a) patru b) cinci c) trei d) șapte

Rezolvare

În primul ciclu contorul i va parcurge pe rând valorile $i=0, 1, 2, 3, 4$. La fiecare pas, se execută un al doilea ciclu, "dirijat" de contorul j , care va lua valori de la 0 la $k-1$, adică $j=0, 1, 2$. Vom urmări succesiunea de pași a acestor două cicluri imbricate, pentru $n=5$ și $k=3$.

Pasul 1: $i=0, j=0, 1, 2$

Pasul 1.1: $j=0, v[i+j]=k+i \Rightarrow v[0+0]=3+0, v[0]=3$

Pasul 1.2: $j=1 \Rightarrow v[0+1]=3+0, v[1]=3$

Pasul 1.3: $j=2 \Rightarrow v[2]=3$

De fapt, observăm deja că, la un pas al primului ciclu, în cadrul celui de-al doilea ciclu toate elementele vor primi aceeași valoare $k+i$, independentă de contorul j .

Absolut analog:

La pasul 2, pentru $i=1$, parcurgând al doilea ciclu cu $j=0, 1, 2$, obținem $v[1]=v[2]=v[3]=4$;

La pasul 3, pentru $i=2$, rezultă $v[2]=v[3]=v[4]=5$;

La pasul 4, pentru $i=3$ obținem $v[3]=v[4]=v[5]=6$;

La pasul 5, pentru $i=4$, se produc ultimele modificări ale elementelor, care conduc la $v[4]=v[5]=v[6]=7$.

Plecând de la configurația inițială a vectorului v cu toate elementele nule, adică $v=(v[0], v[1], v[2], \dots)=(0, 0, 0, 0, 0, 0, \dots)$ (am evidențiat primele șapte elemente care se modifică în cadrul algoritmului), pentru a afla configurația finală vom cerceta în ordine inversă succesiunea de atribuirii care modifică elementele. De ce? Pentru că ultima modificare va da valoarea finală a fiecărui element! Astfel:

– la pasul 5 se stabilesc valorile finale $v[4]=v[5]=v[6]=7$;

– la pasul 4 avem o singură valoare finală, $v[3]=6$, deoarece elementele $v[4]$ și $v[5]$ se vor modifica ulterior, la pasul 5;

– similar, obținem ca valori finale $v[2]=5$ la pasul 3, $v[1]=4$ la pasul 2 și $v[0]=3$ la pasul 1.

Prin urmare, vectorul va fi $v=(3, 4, 5, 6, 7, 7, 7, 0, 0, 0, \dots)$. S-au modificat doar primele șapte elemente, în vreme ce restul au rămas cu valoare inițială 0, deci în final vectorul va conține cinci valori distincte nenule, respectiv 3, 4, 5, 6 și 7.

Răspuns corect: b)

Aplicație

R.V.3. Medie aritmetică într-un șir

Se dă un șir v cu n elemente numere întregi. Scrieți un program care determină și afișează pe ecran media aritmetică a elementelor pare din șir.

Exemplu: Pentru vectorul $v = (-3, 8, 4, -5, 10, 7, 8)$, cu $n = 7$ elemente, programul va afișa valoarea 7.5 (adică valoarea expresiei $(8+4+10+8)/4$)

Rezolvare I:

Memorăm șirul dat într-un vector v cu maxim 25 de elemente numere întregi. În secțiunea de declarații vom declara vectorul împreună cu celelalte variabile: `{int i,n,v[20],s,nr;}`. Dacă n este numărul real de elemente, atunci elementele efectiv folosite vor fi $v[0], v[1], \dots, v[n-1]$.

Citim de la tastatură valoarea lui n , apoi, într-un ciclu, citim elementele vectorului: parcurgem pozițiile elementelor $i=0, 1, \dots, n-1$ și pentru fiecare valoare a lui i , citim elementul $v[i]$:

```
for (i=0; i<=n-1; i++)
{
    cout << endl << "v[" << i << "]=";
    cin >> v[i];
}
```

Fie s și nr suma respectiv numărul elementelor pare din vector, variabile care se inițializează cu 0. Parcurgem într-un ciclu `for` pozițiile elementelor $i=0, 1, \dots, n-1$. Pentru fiecare valoare a lui i , testăm dacă elementul $v[i]$ (de pe poziția i) este par `"v[i]&2==0"`, ceea ce înseamnă `"!(v[i]&2!=0)"`, condiție care se poate scrie simplificat `"!(v[i]&2)"`. În caz afirmativ:

- adăugăm $v[i]$ la suma elementelor pare s `{s+=v[i];}`;
- mărim cu 1 numărul nr al elementelor pare din vector `{nr++;}`.

```
nr=0; s=0;
for (i=0; i<=n-1; i++)
    if (!(v[i]&2))
    {
        s+=v[i]; nr++;
    }
```

În final, dacă există elemente pare în vector ($nr > 0$), afișăm media aritmetică a acestora, care este `(float)s/nr` (suma elementelor pare împărțită la numărul elementelor pare). Verificarea condiției $nr > 0$ este necesară pentru a evita împărțirea în cazul în care nu există nici un element par în vector (variabilele s și nr ar rămâne cu valoarea inițială 0, iar calculatorul nu poate efectua împărțirea $0/0$!). Cuvântul `"float"` plasat înaintea deîmpărțitului s are rolul de a îl converti de la întreg către flotant (în absența acestei conversii explicite, compilatorul ar furniza câtul întreg al împărțirii, fără partea zecimală, din cauză că ambii operanzi sunt întregi).

```
if (nr>0)
    cout << "\n Media= " << (float)s/nr;
```

Prezentăm în continuare programul:

```
# include <iostream.h>

void main ()
{
    int i,n,v[20],s,nr;
    // citește numărul real de elemente n ale vectorului v
    cout << endl << "n="; cin >> n;
    // citește elementele vectorului v într-un ciclu for
    for (i=0; i<=n-1; i++)
    {
        cout << endl << "v[" << i << "]=";
        cin >> v[i];
    }
    // calculează suma S și numărul nr al elementelor pare din vector
    nr=0; s=0;
    for (i=0; i<=n-1; i++)
        if (!(v[i]&2))
        {
            s+=v[i]; nr++;
        }
    // dacă există elemente pare, afișează media aritmetică a lor
    if (nr>0)
        cout << "\n Media= " << (float)s/nr;
}
```

Rezolvare II:

În soluția anterioară s-au folosit două cicluri `for`: primul pentru citirea vectorului, iar al doilea pentru calculul sumei și numărului elementelor pare. Dar nu puteam folosi un singur ciclu? Răspunsul este afirmativ.

După citirea lui n , inițializăm s și nr cu 0. Apoi scriem un ciclu în care contorul i ia pe rând valorile $0, 1, \dots, n-1$, și pentru fiecare valoare a lui i :

- ♦ citim elementul $v[i]$ (de pe poziția i);
- ♦ testăm dacă $v[i]$ este par. În caz afirmativ adăugăm $v[i]$ la suma s a elementelor pare și incrementăm numărul nr al acestora.

Astfel, actualizarea lui s și nr s-a făcut direct în ciclul de citire!

În final, dacă $nr > 0$, afișăm media aritmetică a elementelor pare s/nr , la fel ca în varianta anterioară.

```
# include <iostream.h>
# include <conio.h>

void main ()
{
    int i,n,v[20],s,nr;
    // citește numărul real de elemente n ale vectorului v
    cout << endl << "n="; cin >> n;
```



```
// intr-un ciclu for, citește elementele vectorului v
// și calculează suma S și numărul nr al elementelor pare din vector
nr=0; S=0;
for (i=0; i<=n-1; i++)
{
    // citește un element v[i]
    cout << endl << "v[" << i << "]=";
    cin >> v[i];
    // testează dacă v[i] este par, iar în caz afirmativ actualizează S și nr
    if (!(v[i]%2))
    {
        S+=v[i]; nr++;
    }
}
if (nr>0)
    cout << "\n Media= " << (float)S/nr;
}
```

Încercați singuri!

► 1. Fiind dat un vector v cu n elemente numere întregi, scrieți câte un program care:

- determină suma elementelor pozitive din vector;
- calculează produsul și numărul elementelor negative aflate pe poziții impare în vector;
- afișează numărul elementelor pozitive, numărul elementelor negative și numărul elementelor nule din vector;
- tipărește elementele pare aflate pe poziții divizibile cu 3;

Pentru fiecare dintre programele anterioare, se cer mai multe variante:

V1) folosind cicluri **for**; V2) folosind cicluri **while-do**; V3) folosind cicluri **do-while**.

► 2. (Bacalaureat iulie 2009, varianta 84)

Scrieți un program care citește de la tastatură un număr natural n ($1 \leq n \leq 100$), un șir de câte n numere întregi, cu cel mult cinci cifre fiecare, notat $a_0, a_1, a_2, \dots, a_{n-1}$, apoi un al doilea șir de n numere întregi, cu cel mult cinci cifre fiecare, notat $b_0, b_1, b_2, \dots, b_{n-1}$. Programul construiește în memorie și afișează pe ecran un șir c format din n numere calculate astfel: $c_i = a_i + b_i$, pentru $i=0, 1, \dots, n-1$. Numerele afișate vor fi separate prin câte un spațiu.

Exemplu: pentru $n=4$ și șirurile (2,3,7,8) respectiv (43,3,1,8), se afișează 45 6 8 16.

► 3. (Bacalaureat iulie 2009, varianta 85)

Scrieți un program care citește de la tastatură un număr natural n ($1 \leq n \leq 100$), un șir de câte n numere întregi, cu cel mult 5 cifre fiecare, notat $a_0, a_1, a_2, \dots, a_{n-1}$, apoi un al doilea șir de n numere întregi, cu cel mult 5 cifre

fiecare, notat $b_0, b_1, b_2, \dots, b_{n-1}$. Programul construiește în memorie și afișează pe ecran un șir c format din n numere calculate astfel: $c_i = a_i - b_i$, pentru $i=0, 1, \dots, n-1$. Numerele afișate sunt separate prin câte un spațiu.

Exemplu: pentru $n=4$ și șirurile (2,3,7,8) respectiv (43,3,1,8), se afișează: -41 0 6 0.

Aplicație

R.V.4. Elementele prime ale unui șir

(Bacalaureat iulie 2009, varianta 67)

Se citește de la tastatură un șir de n numere întregi. Să se afișeze elementele prime ale șirului precum și numărul acestora.

Exemplu: pentru șirul (12,17,8,11,5,26), programul va afișa elementele prime 17, 11 și 5, precum și valoarea 3 (care atestă faptul că în șir există trei numere prime în șir).

Rezolvare

Memorăm cele n elemente ale șirului într-un vector $v=(v[0], v[1], \dots, v[n-1])$. Astfel, n va reprezenta numărul real de elemente din vector. Citim mai întâi valoarea lui n , apoi într-un ciclu **for** citim elementele: contorul i va parcurge pe rând pozițiile $0, 1, \dots, n-1$ ale elementelor, și la fiecare pas citim un element $v[i]$.

Vom număra elementele prime ale vectorului într-o variabilă nr , pe care o inițializăm cu 0. Parcurgem într-un alt ciclu **for** pozițiile elementelor $i=0, 1, \dots, n-1$. Pentru fiecare element $v[i]$, dacă este diferit de zero, trebuie să testăm dacă este prim. În caz afirmativ, afișăm $v[i]$ și să incrementăm nr . Algoritmul prin care verificăm dacă $v[i]$ este număr prim nu mai reprezintă o noutate:

- Folosim o variabilă ok , de tip întreg dar cu sens logic, a cărei valoare, 1 sau 0, va indica în final dacă elementul $v[i]$ este prim sau nu. Presupunem mai întâi condiția îndeplinită, inițializând ok cu 1.
- Într-un nou ciclu **for**, prin contorul k vor trece pe rând posibili divizori ai lui $v[i]$. Aceștia sunt numerele naturale de la 2 la "parte întreagă din $v[i]/2$ ": $k=2, 3, \dots, v[i]/2$. Pentru fiecare dintre aceste valori ale lui k , verificăm dacă este efectiv divizor al lui $v[i]$ {" $v[i]\%2==0$ "}, care se poate scrie mai simplu " $!(v[i]\%2)$ ". În caz afirmativ, găsim un divizor al lui $v[i]$ "în persoana" celui k , rezultă că elementul $v[i]$ nu mai poate fi prim, fapt pe care îl marcăm prin atribuirea $ok=0$.

```
ok=1;
for (k=2; k<=v[i]/2; k++)
    if (!(v[i]%k))
        ok=0;
```

- Dacă ok a rămas 1 ("if (ok==1)" se poate scrie prescurtat "if (ok)"), înseamnă că elementul $v[i]$ este prim și:
 - afișăm $v[i]$;
 - incrementăm cu 1 numărul nr al elementelor prime din vector { $nr++$ }.

Observație: Pentru fiecare element $v[i]$, testarea " $v[i] \neq 0$ " este necesară deoarece 0 este "văzut" ca număr prim (prin folosirea operatorului "%").

```
#include <iostream.h>
#include <conio.h>

void main ()
{
    clrscr();
    int v[50], i, k, n, nr, ok;
    cout << endl << "Dati numarul de elemente n: ";
    cin >> n;
    // citeste cele n elemente ale vectorului v
    for (i=0; i<=n-1; i++)
    {
        cout << "Dati elem de pe pozitia " << i << ": ";
        cin >> v[i];
    }
    cout << endl << "Elementele prime sunt: " << endl;
    nr=0; // nr reprezinta numarul elementelor prime
    for (i=0; i<=n-1; i++)
        if (v[i])
        {
            // pentru fiecare element v[i], testam daca este prim
            ok=1;
            for (k=2; k<=v[i]/2; k++)
                if (!(v[i]%k))
                    ok=0;
            // daca v[i] este prim, atunci il afisam si incrementam nr
            if (ok)
            {
                cout << v[i] << " ";
                nr++;
            }
        }
    cout << endl << "In vector sunt " << nr << " elemente prime";
    getch();
}
```

Încercați singuri!

► 1. (Bacalaureat iulie 2008, varianta 71)

Scrieți un program care citește de la tastatură un număr natural n , apoi n numere naturale de cel mult nouă cifre, și afișează pe ecran câte dintre aceste numere au proprietatea că suma cifrelor lor este un număr prim.

Exemplu: pentru $n=8$ și numerele (12, 13, 20, 51, 60, 122, 128, 117), programul va afișa valoarea 3 (reprezentând numărul valorilor subliniate, care îndeplinesc condiția cerută).

► 2. Se citește de la tastatură un număr natural n , apoi un șir cu n elemente numere întregi, cu cel puțin două cifre fiecare. Scrieți un program care afișează acele elemente ale șirului care au prima cifră egală cu ultima cifră.

Exemplu: pentru șirul (2412, 138, 556, 363, 14, 11), programul va afișa numerele 2412, 363 și 11.



V.1.3. Algoritmi fundamentali de prelucrare a tablourilor

Determinarea minimului / maximului dintr-un șir de numere

Considerând un șir de n elemente numere întregi, memorat într-un vector ($v[0], v[1], \dots, v[n-1]$), se pune problema determinării elementului maxim (cel mai mare) și a elementului minim (cel mai mic).

Algoritm ► Determinarea minimului dintr-un șir de numere

Memorăm minimul într-o variabilă min . Presupunem inițial că minimul este cel mai mare număr întreg recunoscut de limbaj, $MAXINT$ (o constantă simbolică predefinită, existentă în header-ul " $values.h$ ", cu valoarea 32767) ($min=MAXINT$). Într-un ciclu, contorul i parcurge pozițiile elementelor $i=0, 1, \dots, n-1$. Pentru fiecare valoare a lui i , comparăm elementul $v[i]$ cu minimul pe care îl avem în acel moment în variabila min . Dacă $v[i]$ este mai mic decât minimul min , atunci elementul respectiv $v[i]$ devine noul minim ($min=v[i]$).

```
min=MAXINT;
for (i=0; i<=n-1; i++)
    if (v[i]<min)
        min=v[i];
```

Exemplu:

Considerăm vectorul $v=(-3, 6, -4, -11, 7)$, cu $n=5$ elemente. Inițial $min=MAXINT=32767$. Urmărim pașii ciclului:

```
i=0: v[0]<min? -3<32767? da => min=v[0] => min=-3;
i=1: v[1]<min? 6<-3? nu
i=2: v[2]<min? -4<-3? da => min=v[2] => min=-4;
i=3: v[3]<min? -11<-4? da => min=v[3] => min=-11;
i=4: v[4]<min? 7<-11? nu
```

În variabila min s-a obținut minimul din tot vectorul, care este -11.

Observăm că, indiferent ce valoare ar avea, primul element $v[1]$ va îndeplini

condiția din linia **if**. Prin urmare, la primul pas al ciclului se va face atribuirea $\text{min} = v[0]$. Consecința? Este posibil ca ulterior minimul min să devină alt element, dar oricum în mod sigur el va fi ales din rândul elementelor vectorului.

Algoritm ► Determinarea maximului dintr-un șir de numere

Se actualizează maximul într-o variabilă **max**. Algoritmul este asemănător cu cel de minim. Inițializăm maximul **max** cu cel mai mic număr întreg recunoscut de limbaj, **-MAXINT**. Parcurgem într-un ciclu pozițiile $i=0, 1, \dots, n-1$ ale elementelor. La fiecare pas, dacă elementul $v[i]$ este mai mare decât maximul **max**, atunci noul maxim va fi $v[i]$.

```
max=-MAXINT;
for (i=0; i<=n-1; i++)
    if (v[i]>max)
        max=v[i];
```

Prezentăm în continuare o altă variantă a algoritmilor de mai sus, în care maximul / minimul se inițializează cu primul element, apoi se actualizează prin parcurgerea vectorului începând cu elementul al doilea. Dacă urmăriți cu atenție pas cu pas secvențele de mai jos, veți vedea că efectul este același!

<pre>{ Minim: } min=v[0]; for (i=1; i<=n-1; i++) if (v[i]<min) min=v[i];</pre>	<pre>{ Maxim: } max=v[0]; for (i=1; i<=n-1; i++) if (v[i]>max) max=v[i];</pre>
--	--

Aplicație R.V.5. Interschimbare între minim și maxim

(Bacalaureat, iulie 2009, varianta 25, enunț adaptat)

Se citesc de la tastatură un număr natural n ($2 \leq n \leq 50$), apoi un tablou unidimensional **v**, care conține n valori naturale. Scrieți un program care interschimbă între ele cel mai mic element par și cel mai mare element impar din tabloul dat, tipărind tabloul obținut. Dacă în tabloul citit nu există nici un element par sau nici un element impar, acesta va rămâne nemodificat, și se va afișa un mesaj de atenționare.

Exemplu: pentru $n=6$ și tabloul (12,3,6,1,24,17), programul va interschimba între ele elementele 6 și 17, iar tabloul obținut va fi (12,3,17,1,24,6).

Rezolvare

■ Mai întâi citim valoarea lui n , reprezentând numărul real de elemente ale vectorului (tabloului unidimensional) **v**, după care citim elementele propriu-zise, folosind

algoritmul prezentat pe larg în partea de teorie: într-un ciclu, contorul i va parcurge pozițiile $0, 1, \dots, n-1$ ale elementelor, și la fiecare pas citim elementul $v[i]$ (aflat pe poziția i în vector)

```
for (i=0; i<=n-1; i++)
{
    cout << "Dati elem de pe pozitia " << i << ": ";
    cin >> v[i];
}
```

■ Definim următoarele variabile:

- **minpar** și **pozpar** = minimul dintre elementele pare ale vectorului, respectiv poziția pe care se află acest minim în vector;

- **maximp** și **pozimp** = maximul dintre elementele impare ale vectorului, respectiv poziția pe care se găsește acest maxim.

De ce avem nevoie și de pozițiile minimului și maximului? Pentru că la final aceste două valori trebuie interschimbate între ele, deci noi avem nevoie să știm pe ce poziții se află (nu uitați că după localizarea acestor elemente, parcurgerea merge mai departe, până la sfârșitul vectorului!).

Pentru a obține un algoritm eficient, cu un număr cât mai mic de operații, vom actualiza cele două valori extreme în paralel, în cadrul aceluiași ciclu. Inițializăm minimul elementelor pare cu cea mai mare valoare întreagă pară posibilă, **MAXINT-1** (atenție, nu cu **MAXINT**, pentru că **MAXINT** are valoarea 32767, care este un număr impar!) {**minpar=MAXINT-1**}. Analog, inițializăm maximul elementelor impare cu cea mai mică valoare întreagă impară posibilă, **-MAXINT** {**maximp=-MAXINT**}.

■ În continuare, parcurgem din nou întreg vectorul **v**. Într-un ciclu, contorul i va lua ca valori pozițiile $0, 1, \dots, n-1$ ale elementelor, și la fiecare pas:

- ✓ dacă elementul $v[i]$ este par ($v[i]\%2==0$, care se poate scrie mai elegant $!(v[i]\%2)$) și totodată este mai mic decât minimul elementelor pare existent în acel moment ($v[i]<\text{minpar}$), atunci:
 - noul minim al elementelor pare devine elementul $v[i]$ {**minpar=v[i]**};
 - poziția **pozpar** a respectivului minim devine momentan chiar i {**pozpar=i**};
- ✓ dacă elementul $v[i]$ este impar ($v[i]\%2!=0$, care se scrie simplu $v[i]\%2$) și totodată este mai mare decât maximul elementelor impare ($v[i]>\text{maximp}$), atunci:
 - noul maxim al elementelor impare devine elementul $v[i]$ {**maximp=v[i]**};
 - poziția **pozimp** a respectivului maxim devine momentan chiar i {**pozimp=i**};

```
minpar=MAXINT-1;
maximp=-MAXINT;
for (i=0; i<=n-1; i++)
{
    if (!(v[i]%2) && v[i]<minpar)
    {
        minpar=v[i]; pozpar=i;
    }
}
```

```

        if (v[i]%2 && v[i]>maximp)
        {
            maximp=v[i];
            pozimp=i;
        }
    }

```

☐ Mai departe, ar trebui interschimbate între ele minimul elementelor pare **minpar** (care este de fapt **v[pozpar]**) cu maximul elementelor impare **maximp** (adică **v[pozimp]**). Numai că există și posibilitatea ca vectorul **v** să nu fi avut nici un element par, sau nici un element impar, caz în care nu avem ce interschimba. În consecință, mai sunt necesare niște testări:

– dacă **minpar** a rămas cu valoarea inițială **MAXINT-1**, înseamnă că în vector nu s-a găsit nici un element par, caz în care afișăm un mesaj de avertizare:

– în caz contrar, dacă **maximp** a rămas cu valoarea inițială **-MAXINT**, rezultă că vectorul nu conține nici un număr impar, deci iarăși trebuie tipărit un mesaj.

– în sfârșit, dacă nu ne încadrăm în nici unul din cazurile particulare de mai sus, înseamnă că avem valori concrete pentru **minpar** și **maximp**, care, așa cum am mai spus, sunt de fapt elementele **v[pozpar]** și **v[pozimp]**.

✓ Acestea trebuie interschimbate cu "metoda paharelor":

- salvăm **v[pozpar]** într-o variabilă temporară **temp**
`{v[pozpar]=temp;};`
- în locul lui **v[pozpar]** aducem elementul **v[pozimp]**, prin atribuirea
`{v[pozpar]=v[pozimp];};`
- în locația lui **v[pozimp]** memorăm valoarea inițială a lui **v[pozpar]**, pe care o avem salvată în **temp** `{v[pozimp]=temp;};`

✓ Odată făcută interschimbarea, nu ne rămâne decât să afișăm vectorul în forma sa finală: într-un ciclu cu $i=0, 1, \dots, n-1$, la fiecare pas scriem pe ecran elementul **v[i]**.

```

        for (i=0; i<=n-1; i++)
            cout << v[i] << " ";

```

☐ Programul complet este următorul:

```

// Bac 2009, II.3 / var 25
#include <iostream.h>
#include <conio.h>
#include <values.h>
void main ()
{
    clrscr();
    int v[50], i, n, maximp, minpar, pozpar, pozimp, temp;
    cout << endl << "Dati numarul de elemente n: ";
    cin >> n;
    // citeste cele n elemente ale vectorului v
    for (i=0; i<=n-1; i++)
    {
        cout << "Dati elem de pe pozitia " << i << ": ";
        cin >> v[i];
    }
    // minpar = minimul elementelor pare, maximp = maximul elem impare

```

```

// pozpar si pozimp = pozitia minimului respectiv maximului in vector
minpar=MAXINT-1;
maximp=-MAXINT;
for (i=0; i<=n-1; i++)
{
    // testeaza daca v[i] devine noul minim dintre elem pare, actualizand minpar si pozitia sa
    if (!(v[i]%2) && v[i]<minpar)
    {
        minpar=v[i]; pozpar=i;
    }
    // testeaza daca v[i] devine noul maxim dintre elem impare, actualizand maximp si pozitia sa
    if (v[i]%2 && v[i]>maximp)
    {
        maximp=v[i]; pozimp=i;
    }
}
if (minpar==MAXINT-1)
    cout << endl << "Vectorul nu are nici un element par!";
else
    if (maximp==MAXINT)
        cout << endl << "Vectorul nu are nici un element impar!";
    else
    {
        // interschimba minimul elementelor pare cu maximul elementelor impare
        cout << endl << "Minimul dintre elem. pare este " << minpar;
        cout << endl << "Maximul dintre elem. impare este " << maximp;
        temp=v[pozpar];
        v[pozpar]=v[pozimp];
        v[pozimp]=temp;
        cout << endl << "Vectorul rezultat dupa interschimbare este: ";
        for (i=0; i<=n-1; i++)
            cout << v[i] << " ";
    }
}

```

Încercați singuri !

► 1. Se citește de la tastatură un șir de **n** numere întregi. Scrieți un program care afișează cea mai mare diferență în modul dintre doi termeni consecutivi ai șirului.

Exemplu: pentru șirul (3, 11, 4, 20, 7, 15), programul va afișa valoarea 16 (adică |20-4|).

► 2. Realizați un program care, pentru un șir de **n** numere întregi citit de la tastatură, determină cele mai mici două elemente ale șirului. Acestea vor fi afișate pe un rând de ecran, în ordine descrescătoare, cu un spațiu între ele.

Exemplu: pentru șirul (5, -8, 11, 3, -9, 22, 47), programul va afișa valorile -8 și -9, în această ordine.

Algoritmi de căutare

Algoritm ► Căutarea unei valori într-un vector

Fiind dat un vector v cu n elemente numere întregi, se pune problema de a stabili dacă o valoare dată x se găsește sau nu printre elementele vectorului.

Vom folosi o variabilă `gasit` de tipul `int`, a cărei valoare va simula valoarea de adevăr a condiției studiate: 1 dacă x există în vector, respectiv 0 în caz contrar. Astfel de variabile se întâlnesc frecvent în algoritmi informatici, mulți programatori denumindu-le sugestiv "variabile-semafor".

Folosim metoda reducerii la absurd. Înainte de a începe căutarea, presupunem că condiția testată este falsă, adică valoarea x nu există în vector, deci inițializăm `gasit` cu 0 (valoare care corespunde stării de "condiție falsă"). Parcurgem într-un ciclu pozițiile elementelor, $i=0, 1, \dots, n-1$. Pentru fiecare valoare a lui i , testăm dacă elementul $v[i]$ este egal cu x : în caz afirmativ variabila `gasit` devine 1 (marcând astfel faptul că valoarea x a fost găsită în vector, pe poziția i , adică faptul că acum condiția în discuție a devenit adevărată).

```
gasit=0;
for (i=0; i<=n-1; i++)
    if (v[i]==x)
        gasit=1;
```

Exemplu:

Fie vectorul $v=(2, 6, -1, 4, 7)$, cu $n=5$ elemente. Urmărim funcționarea algoritmului în cazul în care valoarea căutată este $x=-1$:

Inițial `gasit=0` și în ciclu $i=0, 1, 2, 3, 4$;

```
i=0: v[0]==x? 2== -1? nu
i=1: v[1]==x? 6== -1? nu
i=2: v[2]==x? -1== -1? da => gasit=1
i=3: v[3]==x? 4== -1? nu
i=4: v[4]==x? 7== -1? nu
```

Deci la pasul al treilea al ciclului variabila `gasit` a devenit 1, marcând astfel faptul că valoarea $x=-1$ se găsește pe poziția $i=2$.

Dacă valoarea căutată ar fi fost de exemplu $x=5$, atunci pentru nici o valoare a lui i nu s-ar fi îndeplinit condiția " $v[i]==x$ " din linia `if`. În consecință, valoarea variabilei `gasit` ar fi rămas cea inițială, adică 0.

Observație:

O greșeală frecvent întâlnită este următoarea variantă a algoritmului de căutare:

```
gasit=0;
for (i=0; i<=n-1; i++)
    if (v[i]==x)
        gasit=1;
    else
        gasit=0;
```

Pentru a înțelege de ce este greșit, să reluăm exemplul anterior: vectorul $v=(2, 6, -1, 4, 7)$ cu $n=5$ elemente și valoarea căutată $x=-1$.

Inițial `gasit=0` și în ciclu $i=0, 1, \dots, n-1$;

```
i=0: v[1]==x? 2== -1? nu => execută ramura else => gasit=0;
i=1: v[1]==x? 6== -1? nu => execută ramura else => gasit=0;
i=2: v[2]==x? -1== -1? da => execută ramura if => gasit=1;
```

În continuare valoarea variabilei `gasit` ar trebui să rămână 1, deoarece această valoare marchează faptul că x există în vector. Dar iată ce se va întâmpla la pasul următor al ciclului:

```
i=3: v[3]==x? 4== -1? nu => execută ramura else => gasit=0;
```

Așadar variabila `gasit` a redevenit 0, cu toate că valoarea căutată x există în vector ("în persoana" lui $v[2]$). Aceeași atribuire `gasit=0` se va face și la ultimul pas, deci la ieșirea din ciclu valoarea variabilei `gasit` va fi 0, cu toate că x există în vector. Care trebuia să fie valoarea variabilei `gasit`? Păi 1 dacă x există în vector, respectiv 0 în caz contrar. Se vede clar că nu e așa!

Aplicație R.V.6. Intersecția a două mulțimi

Se dau două mulțimi cu câte n elemente fiecare, definite ca vectori. Să se afișeze intersecția celor două mulțimi (elementele lor comune).

Exemplu: pentru mulțimile $A=\{2, 3, 5, 7, 11\}$ și $B=\{3, 7, 9, 10, 12\}$, memorate în vectorii a și b cu câte $n=5$ elemente fiecare, programul va afișa valorile 3 și 7, reprezentând elementele comune celor două mulțimi.

Rezolvare

 Declarăm cele două mulțimi a și b ca vectori cu maxim 20 de elemente fiecare. Citim numărul efectiv de elemente al celor doi vectori, în variabila n . Cei doi vectori vor fi citiți "în paralel". Parcurgem într-un ciclu pozițiile $i=0, 1, \dots, n-1$ ale elementelor în cei doi vectori, și la fiecare pas citim atât elementul $a[i]$, cât și elementul $b[i]$.

```
cout << endl << "n=";
cin >> n;
for (i=0; i<=n-1; i++)
{
    cout << "a[" << i << "]="; cin >> a[i];
    cout << "b[" << i << "]="; cin >> b[i];
}
```

Urmează determinarea intersecției propriu-zise a celor două mulțimi memorate acum în vectorii *a* și *b*. Parcurgem într-un ciclu pozițiile $i=0, 1, \dots, n-1$ ale elementelor în primul vector și la fiecare pas:

- verificăm dacă elementul $a[i]$ se găsește și în cel de-al doilea vector *b*, folosind algoritmul de căutare descris anterior. În urma căutării, variabila booleană *gasit* va fi: 1 dacă $a[i]$ se află în vectorul *b*, respectiv 0 în caz contrar;

```
for (i=0; i<=n-1; i++)
{
    gasit=0;
    for (j=0; j<=n-1; j++)
        if (b[j]==a[i])
            gasit=1;
```

- testăm variabila *gasit* și dacă valoarea sa este 1, înseamnă că $a[i]$ este un element comun celor doi vectori (adică un element al intersecției mulțimilor identificate de către vectori), deci îl afișăm pe patru caractere.

```
if (gasit)
    cout << a[i] << " " ;
```

```
# include <iostream.h>

void main ()
{
    int i,j,m,n,gasit,a[20],b[20];
    cout << endl << "n="; cin >> n;
    // citește într-un ciclu vectorii a și b, cu m respectiv n elemente
    for (i=0; i<=m-1; i++)
    {
        cout << "a[" << i << "]="; cin >> a[i];
        cout << "b[" << i << "]="; cin >> b[i];
    }
    cout << endl;
    // parcurgem pozițiile i=0,1,...,n-1 ale elementelor vectorului a
    for (i=0; i<=n-1; i++)
    {
        // la fiecare pas, verificăm dacă elementul a[i] al vectorului a, se găsește și în vectorul b
        gasit=0;
        for (j=0; j<=n-1; j++)
            if (b[j]==a[i])
                gasit=1;
        // dacă a[i] se afla și în b, atunci îl afișăm ca și element al intersecției
        if (gasit) cout << a[i] << " " ;
    }
}
```

Încercați singuri !

►1. Scrieți un program care verifică dacă o valoare *x* se găsește sau nu printre cele *n* elemente ale unui șir dat. În caz afirmativ se va afișa poziția / pozițiile valorii *x* în vector, iar în caz contrar se va tipări un mesaj de eroare.

Valorile lui *n* și *x*, precum și elementele șirului se citesc de la tastatură.

Exemplu: pentru șirul (7, -32, 4, 7, -11, 8, -2, 7) și *x*=7, programul va afișa 0, 3 și 7, adică pozițiile pe care apare numărul 7 ca element în vector.

►2. Se dau două mulțimi definite prin intermediul vectorilor *u* și *v*, cu *m* respectiv *n* elemente. Să se memoreze în vectorul *w* și apoi să se afișeze diferența celor două mulțimi (elementele care se găsesc în prima și nu se găsesc în cea de-a doua).

Exemplu: pentru mulțimile $A=\{2,3,5,7,11\}$ și $B=\{3,7,9,10,12\}$, memorate în vectorii *a* și *b* cu câte *n*=5 elemente fiecare, programul va afișa valorile 3 și 7, reprezentând elementele comune celor două mulțimi.

Algoritmi de formare a perechilor de elemente într-un vector

Aplicație R.V.7. Perechi de elemente într-un șir

Se citește de la tastatură un vector *v* cu *n* elemente numere întregi. Să se tipărească toate perechile de elemente cu proprietatea că cel mai mare divizor comun al elementelor perechii este o valoare dată *d*.

Exemplu: dacă vectorul este $v=(2,3,6,5,8)$ cu *n*=5 elemente, atunci:

- pentru *d*=2, perechile căutate sunt (2, 6), (2, 8), (6, 8);
- pentru *d*=3, există o singură astfel de pereche, și anume (3, 6).

Rezolvare

Citim mai întâi valoarea lui *n*, apoi într-un ciclu *for* citim elementele vectorului: contorul *i* va parcurge pe rând pozițiile 0, 1, ..., *n*-1 ale elementelor, și la fiecare pas citim un element $v[i]$.

Pentru a forma perechile de elemente din vector (atenție, nu neapărat consecutive!), vom alcătui perechi cu indicii (pozițiile) elementelor. Acestea din urmă sunt perechile distincte de forma (i, j), cu $i, j \in \{0, 1, \dots, n-1\}$. Fiecărei perechi de indici (i, j) îi va corespunde firește perechea de elemente ($v[i], v[j]$).

Să luăm ca exemplu un vector cu *n*=5 elemente. Perechile distincte pe mulțimea de indici {0, 1, 2, 3, 4} sunt

```
(0,1), (0,2), (0,3), (0,4)
(1,2), (1,3), (1,4)
(2,3), (2,4)
(3,4)
```

De notat că, dacă am luat perechea (0,1), atunci nu vom mai lua și perechea (1,0) pentru că de fapt este același lucru.

Ca să scriem aceste perechi sub forma (i, j) trebuie să observăm că:

- primul indice al perechii, adică i , evoluează de la 0 la 3, adică pe caz general i ia valori de la 0 la $n-2$.
- dacă primul indice i este 0 atunci al doilea indice j ia valorile 1, 2, 3, 4, apoi dacă i este 1 atunci j ia valorile 2, 3, 4, etc. Așadar, pe caz general, j va lua valori de la $i+1$ la $n-1$.

În două cicluri, cu $i=0, 1, \dots, n-2$ și $j=i+1, \dots, n-1$, vom forma perechile de indici (i, j) , așa cum am arătat mai sus. Pentru fiecare pereche (i, j) :

♦ Trebuie determinat cel mai mare divizor comun al elementelor corespunzătoare $v[i]$ și $v[j]$. Copiem elementele $v[i]$ și $v[j]$ în două variabile, a respectiv b . În loc să calculăm direct c.m.m.d.c. al lui $v[i]$ și $v[j]$ îl vom afla pe cel al copiilor a și b . Folosim algoritmul lui Euclid cu diferențe, care a fost prezentat într-un capitol anterior.

```
a=v[i]; b=v[j];
while (a!=b)
    if (a>b)
        a-=b; // adică a=a-b
    else
        b-=a; // adică b=b-a
```

♦ După încheierea ciclului de mai sus, când a și b au devenit egale, valoarea lor comună reprezintă c.m.m.d.c. al lui a și b , respectiv c.m.m.d.c. al lui $v[i]$ și $v[j]$. Dacă acesta este egal cu valoarea dată d , înseamnă că am găsit o pereche $(v[i], v[j])$ care îndeplinește proprietatea cerută, deci o afișăm.

Observație: Folosirea unor copii ale elementelor $v[i]$ și $v[j]$ în algoritmul de calcul al lui c.m.m.d.c. este obligatorie. Ce s-ar fi întâmplat dacă am fi aplicat direct algoritmul elementelor $v[i]$ și $v[j]$? Acestea s-ar fi modificat prin scăderi repetate. Dar atât $v[i]$ cât și $v[j]$ mai pot să facă parte și din alte perechi, care urmează a fi testate la pașii următori ai ciclului, deci nu este permisă modificarea valorilor lor!

```
#include <iostream.h>
#include <conio.h>

void main ()
{
    clrscr();
    int v[50], i, j, n, d, a, b;
    cout << endl << "Dati numarul de elemente n: ";
    cin >> n;
    // citeste cele n elemente ale vectorului v
    for (i=0; i<=n-1; i++)
    {
        cout << "Dati elem de pe pozitia " << i << ": "; cin >> v[i];
    }
    cout << endl << "d="; cin >> d;
    // formam perechile de elemente de forma (v[i], v[j]) cu i=1,...,n-1 si j=i+1,...,n
    cout << "Perechile cerute sunt:" << endl;
    for (i=0; i<=n-2; i++)
        for (j=i+1; j<=n-1; j++)
        {
```

```
// copiem elementele perechii (v[i], v[j]) in variabilele a si b
// determinam cel mai mare divizor comun al lui a si b, respectiv v[i] si v[j]
a=v[i]; b=v[j];
while (a!=b)
    if (a>b)
        a-=b;
    else
        b-=a;
// cel mai mare divizor comun al lui v[i] si v[j] este a (sau b)
// verificam daca este egal cu valoarea data d, daca da, afisam perechea (v[i], v[j])
if (a==d)
    cout << v[i] << " " << v[j] << endl;
}
```

Încercați singuri!

► 1. Se citește de la tastatură un vector v cu n elemente numere întregi. Să se tipărească toate perechile de elemente din vector cu proprietatea că al doilea element al perechii este divizibil cu suma cifrelor primului.

Exemplu: pentru vectorul $v=(235, 20, 47, 30, 57, 33)$, perechile cerute sunt $(235, 20)$, $(235, 30)$, $(20, 30)$, $(47, 33)$, $(30, 33)$ și $(47, 33)$.

► 2. (Bacalaureat iulie 2009, varianta 64 – enunț adaptat)

Se citește de la tastatură un tablou unidimensional $a=(a_0, a_1, \dots, a_{n-1})$, cu n elemente numere întregi. Scrieți un program care afișează numărul de perechi (a_i, a_j) , $0 \leq i < j \leq n-1$, în care a_i este divizor al lui a_j , sau a_j este divizor al lui a_i .

Exemplu: pentru $n=5$ și $a=(4, 8, 3, 9, 4)$ programul va afișa valoarea 4, care numără perechile $(4, 4)$, $(4, 8)$, $(8, 4)$ și $(3, 9)$.

Algoritmi de sortare și interclasare

Algoritm ► Sortarea unui vector prin metoda interschimbărilor directe

Această tehnică de sortare vă este cunoscută încă din clasa a IX-a, motiv pentru care o reamintim foarte sumar. Considerând un și memorat în vectorul $x=(x[0], x[1], \dots, x[n-1])$, algoritmul de sortare prin interschimbare directă este următorul:

Fiecare din elementele $x[0], x[1], \dots, x[n-2]$ va juca pe rând rolul unui așa-numit pivot. Parcurgem pivoții într-un ciclu în care contorul i ia pe rând valorile $0, 1, \dots, n-2$ și la fiecare pas al ciclului:

– Comparăm pivotul $x[i]$ cu toate elementele aflate după el, $x[i+1], \dots, x[n-1]$. Parcurgem pozițiile acestor elemente într-un alt ciclu cu contorul $j=i+1, \dots, n-1$, și, pentru fiecare element $x[j]$, dacă este mai mic decât pivotul $x[i]$, atunci interschimbăm

$x[j]$ cu $x[i]$ (pentru că, dacă $x[j]$ este mai mic decât $x[i]$, atunci $x[j]$ trebuie să fie înaintea lui $x[i]$ în vectorul sortat).

Interschimbarea lui $x[i]$ cu $x[j]$ se face prin metoda paharelor: salvăm mai întâi $x[i]$ într-o variabilă $temp$, $\{temp=x[i];\}$, apoi aducem în $x[i]$ valoarea lui $x[j]$ $\{x[i]=x[j];\}$ și în sfârșit, memorăm în $x[j]$ valoarea inițială a lui $x[i]$ luată din $temp$ $\{x[j]=temp;\}$.

Când un pivot $x[i]$ și-a încheiat rolul, partea din vector până la elementul $x[i]$ inclusiv, este sortată crescător. După ce toate elementele $x[0], x[1], \dots, x[n-2]$ au fost pivoți, vectorul este în întregime sortat.

```
for (i=0; i<=n-2; i++)
  for (j=i+1; j<=n-1; j++)
    if (x[j]<x[i])
    {
      temp=x[i]; x[i]=x[j]; x[j]=temp;
    }
```

Să urmărim funcționarea algoritmului pentru vectorul $x=(4, -3, 5, -6, 1)$ cu $n=5$ elemente:

0	1	2	3	4
4	-3	5	-6	1

vectorul de dinaintea execuției secvenței

♦ $i=0, x[0]$ pivot

• $j=1, x[1]<x[0] ? -3<4 ? da \Rightarrow x[1] \leftrightarrow x[0]$

0	1	2	3	4
-3	4	5	-6	1

• $j=2, x[2]<x[0] ? 5<-3 ? nu$

• $j=3, x[3]<x[0] ? -6<-3 ? da \Rightarrow x[3] \leftrightarrow x[0]$

0	1	2	3	4
-6	4	5	-3	1

• $j=4, x[4]<x[0] ? 1<-6 ? nu$

♦ $i=1, x[1]$ pivot

• $j=2, x[2]<x[1] ? 5<4 ? nu$

• $j=3, x[3]<x[1] ? -3<4 ? da \Rightarrow x[3] \leftrightarrow x[1]$

0	1	2	3	4
-6	-3	5	4	1

• $j=4, x[4]<x[1] ? 1<-3 ? nu$

♦ $i=2, x[2]$ pivot

• $j=3, x[3]<x[2] ? 4<5 ? da \Rightarrow x[3] \leftrightarrow x[2]$

0	1	2	3	4
-6	-3	4	5	1

• $j=4, x[4]<x[2] ? 1<4 ? da \Rightarrow x[4] \leftrightarrow x[2]$

0	1	2	3	4
-6	-3	1	5	4

♦ $i=3, x[3]$ pivot

• $j=4, x[4]<x[3] ? 4<5 ? da \Rightarrow x[4] \leftrightarrow x[3]$

0	1	2	3	4
-6	-3	1	4	5

Vectorul sortat $x=(x[0], x[1], \dots, x[4])$ va conține elementele $(-6, -3, 1, 4, 5)$, în această ordine.

Algoritm ► Sortarea unui vector prin metoda bulelor

Ideea care stă la baza metodei constă în formarea de perechi de elemente succesive în vectorul dat. Elementele fiecărei perechi vor fi ordonate crescător, iar prin mai multe parcurgeri ale vectorului se va obține un șir în întregime sortat. Desigur, deocamdată exprimarea este generală astfel încât vom ilustra metoda referindu-ne la următorul șir de elemente memorate într-un vector x .

Exemplu:

0	1	2	3
3	-4	8	-5

La prima parcurgere formăm toate perechile posibile de elemente consecutive: $(x[0], x[1]), (x[1], x[2]), \dots, (x[n-2], x[n-1])$. Pe caz general, ele sunt de forma $(x[i], x[i+1])$, cu $i=0, 1, \dots, n-2$.

În exemplul nostru aceste perechi sunt: $(x[0], x[1]), (x[1], x[2]), (x[2], x[3])$.

Ordonăm prin comparație și eventuală interschimbare fiecare pereche $(x[i], x[i+1])$. Să urmărim această parcurgere pe exemplul dat:

• $i=0, 1, \dots, n-2$. deci $i=0, 1, 2$; formăm perechile $(x[i], x[i+1])$ și pentru fiecare testăm dacă $x[i]<x[i+1]$, iar în caz contrar (în speță dacă $x[i]>x[i+1]$) interschimbăm între ele $x[i]$ cu $x[i+1]$.

0	1	2	3
3	-4	8	-5

$i=0 \Rightarrow$ perechea $(x[0], x[1])$; $x[0]<x[1] ? 3<-4 ? nu \Rightarrow x[0] \leftrightarrow x[1]$

0	1	2	3
-4	3	8	-5

$i=1 \Rightarrow$ perechea $(x[1], x[2])$; $x[1]<x[2] ? 3<8 ? da$

$i=2 \Rightarrow$ perechea $(x[2], x[3])$; $x[2]<x[3] ? 8<-5 ? nu \Rightarrow x[2] \leftrightarrow x[3]$

0	1	2	3
-4	3	-5	8

Observăm că după această primă parcurgere vectorul nu este sortat, dar pe ultima poziție a ajuns elementul 8, care este cel mai mare din tot vectorul. În continuare va trebui să mai facem o parcurgere, într-o manieră similară, în urma căreia pe penultima poziție va fi adus elementul cel mai mare dintre cele rămase, adică al doilea ca mărime din vector. Analog, după a treia parcurgere vom aduce pe ante-penultima poziție al treilea element ca mărime, ș.a.m.d. Acest mod de lucru a dat denumirea metodei: așa cum bulele de aer dintr-un vas cu apă se ridică la suprafață, la fel elementele "migreză" către capătul din dreapta al vectorului, în ordinea mărimilor lor.

Credem că procedeul va fi complet înțeles după descrierea detaliată a celei de-a doua parcurgeri. Desigur, a doua parcurgere se va face numai până la

penultimul element deoarece ultimul este maximul dintre elementele vectorului (găsindu-și deja locul pe care trebuie să-l ocupe în vectorul sortat).

- Formăm perechile $(x[i], x[i+1])$, cu $i=0, 1, \dots, n-3$, adică $i=0, 1$.

$i=0 \Rightarrow$ perechea $(x[0], x[1])$; $x[0] < x[1] ? -4 < 3 ? da$

$i=1 \Rightarrow$ perechea $(x[1], x[2])$; $x[1] < x[2] ? 3 < -5 ? nu \Rightarrow x[1] \leftrightarrow x[2]$

0	1	2	3
-4	-5	3	8

În sfârșit, la a treia parcurgere, mergem doar până la ante-penultimul element, deoarece ultimele două elemente și-au găsit pozițiile în urma parcurgerilor de până acum.

- Avem deci perechile $(x[i], x[i+1])$, cu $i=0, 1, \dots, n-4$, adică $i=0$.

$i=0 \Rightarrow$ perechea $(x[0], x[1])$; $x[0] < x[1] ? -4 < -5 ? nu \Rightarrow x[0] \leftrightarrow x[1]$

0	1	2	3
-5	-4	3	8

Acum vectorul este sortat crescător.

În continuare trecem la conceperea unui program care realizează sortarea unui vector folosind metoda bulelor.

În primul rând vom observa că fiecare parcurgere se face într-un vector "mai scurt" cu o poziție. Acest fapt sugerează utilizarea unui ciclu descrescător. Notând cu f poziția finală, până la care parcurgem vectorul, observăm că: la prima parcurgere mergem până la sfârșitul vectorului deci $f=n-1$, la a doua parcurgere ne oprim la penultimul element deci $f=n-2$, ș.a.m.d., ultima parcurgere "vizitând" numai primele două elemente ($f=1$). Proiectând un ciclu **for** care să repete parcurgerile, poziția finală f evoluează descrescător de la $n-1$ la 1.

- La fiecare pas al acestui ciclu are loc o parcurgere a vectorului $(x[0], x[1], \dots, x[f-1])$, care se face într-un alt ciclu, ca în ilustrările anterioare: contorul i ia valori de la 0 la $f-1$ și la fiecare pas formăm perechi de elemente succesive $(x[i], x[i+1])$, pe care le ordonăm crescător:

```
for (i=0; i<=f-1; i++)
    if (x[i]>x[i+1])
    {
        t=x[i]; x[i]=x[i+1]; x[i+1]=t;
    }
```

Programul complet este prezentat în continuare:

```
#include <iostream.h>
#include <conio.h>
int x[20], i, n, f, t;

void main ()
{
```

```
// citește numărul de elemente n și elementele vectorului x
cout << "\n n="; cin >> n;
for (i=0; i<=n-1; i++)
{
    cout << "x[" << i << "]="; cin >> x[i];
}
// sortează crescător vectorul prin metoda bulelor
for (f=n-1; f>=1; f--)
    for (i=0; i<=f-1; i++)
        if (x[i]>x[i+1])
        {
            t=x[i]; x[i]=x[i+1]; x[i+1]=t;
        }
// afișează vectorul sortat
cout << "\n Sirul sortat este: \n";
for (i=0; i<=n-1; i++) cout << x[i] << " ";
getch();
}
```

Acest program poate fi optimizat ! Pentru aceasta observăm că este posibil ca la o anumită parcurgere să nu se facă nici o interschimbare ceea ce nu spune altceva decât faptul că vectorul este deja sortat. Cu alte cuvinte este posibil ca ordinea finală să fie obținută înainte de ultima parcurgere.

Pentru a vă convinge de acest lucru, urmăriți pașii algoritmului pe vectorul $x=(2, 3, -5, 9, 4)$, și veți constata că:

- după prima parcurgere până la poziția $f=n-1=4$ obțineți vectorul $(2, -5, 3, 4, 9)$;
- după a doua parcurgere până la poziția $f=3$ rezultă vectorul deja sortat $(-5, 2, 3, 4, 9)$;

Observația poate fi materializată astfel:

Parcurgerea prin variabila f se va face printr-un ciclu **do-while** având drept condiție finală faptul că nu s-a făcut nici o interschimbare. Vom folosi așadar o variabilă logică **gata** care, la fiecare parcurgere, va avea valoarea:

- 0 dacă s-a efectuat cel puțin o interschimbare (adică a fost detectată o situație în care ordinea naturală este falsă !);
- 1 dacă în cadrul parcurgerii respective nu s-au mai efectuat interschimbări (algoritmul este gata, după cum sugerează numele variabilei !).

Firește contorul f va evolua în ciclul **do-while** tot de la $n-1$ la 1. Inițializăm f cu prima valoare $n-1$. La fiecare pas se execută o parcurgere astfel:

- inițializăm variabila **gata** cu 1, presupunând că în cadrul parcurgerii respective nu vor fi interschimbări;

- facem parcurgerea propriu-zisă: formăm într-un ciclu **for** perechi de forma $(x[i], x[i+1])$, cu $i=0, 1, \dots, f-1$ (așa cum s-a arătat anterior); pentru fiecare astfel de pereche, dacă $x[i]>x[i+1]$, atunci interschimbăm $x[i]$ cu $x[i+1]$ și marcăm faptul că s-a făcut o interschimbare dând valoarea 0 variabilei **gata**.

```
f=n-1;
do
{
```

```

gata=1;
for (i=0; i<=f-1; i++)
    if (x[i]>x[i+1])
    {
        gata=0; t=x[i]; x[i]=x[i+1]; x[i+1]=t;
    }
    f--;
} while (!gata);

```

În linia **while** se testează valoarea variabilei **gata** și în funcție de aceasta se decide dacă se va relua ciclul cu o nouă parcurgere:

- dacă la parcurgerea făcută am avut cel puțin o interschimbare, variabila **gata** a devenit 0;
- dacă nu a existat nici o interschimbare, atunci **gata** a rămas 1 și în consecință, nemaifiind posibile interschimbări, se abandonează ciclul.

Acum putem scrie algoritmul de sortare complet:

```

#include <iostream.h>
#include <conio.h>

int x[20], i, n, f, t, gata;

void main ()
{
    clrscr();
    // citește numărul de elemente n și elementele vectorului x
    cout << "\n n=";
    cin >> n;
    for (i=0; i<=n-1; i++)
    {
        cout << "x[" << i << "]=";
        cin >> x[i];
    }
    // sortează crescător vectorul prin metoda bulelor
    f=n-1;
    do
    {
        gata=1;
        for (i=0; i<=f-1; i++)
            if (x[i]>x[i+1])
            {
                gata=0;
                t=x[i];
                x[i]=x[i+1];
                x[i+1]=t;
            }
        f--;
    } while (!gata);
    // afișează vectorul sortat
    cout << "\n Sirul sortat este: \n";
    for (i=0; i<=n-1; i++)
        cout << x[i] << " ";
}

```

Algoritm ► Interclasarea a doi vectori sortați

Se pune problema ca, pornind de la doi vectori **x** și **y** ordonați crescător, să formăm un vector **z** care să conțină elementele celor doi vectori dați tot în ordine crescătoare. De exemplu:

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	6	7

Vectorul **z** va fi completat pas cu pas. Notăm prin **k** poziția ultimului element în **z**; inițial **k=0** (evident primul element care va fi adăugat în **z** va veni pe poziția 0). Folosim doi indici cu care parcurgem vectorii dați, respectiv **i** pentru **x** și **j** pentru **y**. Evident că inițial **i=0** și **j=0**.

Într-un ciclu, la fiecare pas, comparăm un element **x[i]**, cu un element **y[j]**. Cel mai mic dintre aceste două elemente va fi adăugat la sfârșitul vectorului **z**, iar în vectorul din care provine (**x** sau **y**) vom muta indicele (**i** sau **j**) cu o poziție mai la dreapta. Concret:

- dacă **x[i] ≤ y[j]** atunci adăugăm pe **x[i]** la sfârșitul vectorului **z** pe poziția dată de valoarea curentă a lui **k** (prin atribuirea **z[k]=x[i]**), apoi creăm o nouă poziție la sfârșitul lui **z** prin incrementarea lui **k**, și în sfârșit mutăm pe **i** prin atribuirea **i=i+1**;

- în caz contrar adăugăm pe **y[j]** la sfârșitul vectorului **z** (**z[k]=y[j]**), creăm noua poziție **k** prin incrementarea **k++** apoi mutăm pe **j** prin atribuirea **j=j+1**;

Ciclul se repetă până când unul dintre indici, **i** sau **j**, ajunge la sfârșitul vectorului său (cât timp **i ≤ m-1** și **j ≤ n-1**).

```

i=j=k=0;
while (i<=m-1 && j<=n-1)
    if (x[i]<y[j])
    {
        z[k]=x[i]; k++; i++;
    }
    else
    {
        z[k]=y[j];
        k++;
        j++;
    }
}

```

Vom urmări câțiva pași ai algoritmului descris pentru vectorii **x** și **y** de mai sus. Pentru început **i=0**, **j=0**, **k=0**:

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0								

$y[0] < x[0] \Rightarrow z[0] \leftarrow y[0]$, „avansează” $j=1$, și $k=1$, iar i rămâne 0;

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0								

$x[0] \leq y[1] \Rightarrow z[1] \leftarrow x[0]$, „avansează” $i=1$, și $k=2$, iar j rămâne 1;

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0	1							

$y[1] \leq x[1] \Rightarrow z[2] \leftarrow y[1]$, „avansează” $j=2$, și $k=3$, iar i rămâne 1;

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0	1	2						

$x[1] \leq y[2] \Rightarrow z[3] \leftarrow x[1]$, „avansează” $i=2$, și $k=4$, iar j rămâne 2;

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0	1	2	3					

$x[2] \leq y[2] \Rightarrow z[4] \leftarrow x[2]$, „avansează” $i=3$, și $k=5$, iar j rămâne 2;

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0	1	2	3	4				

$y[2] \leq x[3] \Rightarrow z[5] \leftarrow y[2]$, „avansează” $j=3$, și $k=6$, iar i rămâne 3;

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5			

$x[3] \leq y[3] \Rightarrow z[6] \leftarrow x[3]$, „avansează” $i=4$, și $k=7$, iar j rămâne 3;

x:

0	1	2	3
1	3	4	6

 y:

0	1	2	3	4
0	2	5	6	7

 z:

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6		

Am ajuns la sfârșitul vectorului x , dar observăm că în vectorul y au rămas ne-parcurs elementele $y[4]$ și $y[5]$. Pe de altă parte, se poate observa cu ușurință faptul că ultima valoare la care a ajuns variabila k reprezintă numărul elementelor adăugate până în acest moment în vectorul z (7 elemente în exemplul nostru).

Așadar, după operația descrisă până acum, într-unul din vectorii u și v au rămas niște elemente ne-parcurs. Acestea sunt ordonate crescător, deci pot fi adăugate "în corpore" la sfârșitul vectorului z . Ne putem da seama care este vectorul ne-parcurs în totalitate urmărind valorile la care au ajuns contoarele i și j : dacă $i \leq m$ au rămas ne-parcurs elemente în vectorul x , iar dacă $j \leq n$ este incompletă parcurgerea lui y . În final, porțiunea de vector ne-parcursă se copiază la sfârșitul vectorului z .

```
if (i <= m-1)
    while (i <= m-1)
    { z[k] = x[i]; k++; i++; }
if (j <= n-1)
    while (j <= n-1)
    { z[k] = y[j]; k++; j++; }
```

Acum ultima valoare a lui k reprezintă numărul final de elemente ale vectorului z după încheierea algoritmului. Prin urmare, pozițiile elementelor în

vectorul z pot fi parcurse cu un ciclu în care contorul va lua valorile $0, 1, \dots, k-1$.

Programul propriu-zis începe prin citirea numărului de elemente din cei doi vectori, m respectiv n . La citirea vectorului x cu m elemente, asigurăm introducerea acestora în ordine crescătoare, deci fiecare element $x[i]$ citit trebuie validat, în sensul că este necesar să fie mai mare decât elementul precedent $x[i-1]$. Analog se citește vectorul y cu n elemente.

În continuare interclasăm vectorii x și y . În vectorul z vom avea rezultatul interclasării, iar variabila k reține numărul de elemente al vectorului z .

Programul se încheie cu afișarea vectorului z .

```
#include <iostream.h>
#include <conio.h>

int x[30], y[30], z[30], i, j, k, m, n;

void main ()
{
    cout << "\n m="; cin >> m; // citește m si primul element al vectorului x
    cout << "\nx[0]="; cin >> x[0];
    // citește elementele vectorului x începând cu al doilea, în ordine crescătoare
    for (i=1; i<=m-1; i++)
    do
    {
        cout << "x[" << i << "]="; cin >> x[i];
    } while (x[i] <= x[i-1]);
    cout << "\n n="; cin >> n; // citește n si primul element al vectorului y
    cout << "\ny[0]="; cin >> y[0];
    // citește elementele vectorului y începând cu al doilea
    for (i=1; i<=n-1; i++)
    do
    {
        cout << "y[" << i << "]="; cin >> y[i];
    } while (y[i] <= y[i-1]);
    // interclasează vectorii x și y
    i=j=k=0;
    while (i<=m-1 && j<=n-1)
    if (x[i] < y[j])
    { z[k] = x[i]; k++; i++; }
    else
    { z[k] = y[j]; k++; j++; }
    // adaugă elementele rămase ne-parcurs într-unul din vectori
    if (i<=m-1)
    while (i<=m-1) { z[k] = x[i]; k++; i++; }
    if (j<=n-1)
    while (j<=n-1) { z[k] = y[j]; k++; j++; }
    // afișează vectorul rezultat după interclasare
    cout << "\n Vectorul sortat este: \n";
    for (i=0; i<=k-1; i++) cout << z[i] << " ";
}
```

Încercați singuri!

►1. Se dă un vector v cu n elemente numere întregi. Scrieți câte un program care ordonează *descrescător* vectorul, folosind:

- algoritmul de sortare prin interschimbare directă;
- algoritmul de sortare prin metoda bulelor.

►2. Se citesc de la tastatură doi vectori u și v cu m respectiv n elemente, *sortați descrescător*. Folosind algoritmul de interclasare, să se formeze un vector w , conținând elementele celor doi vectori aranjate în ordine descrescătoare.

►3. (Bacalaureat iulie 2008, varianta 59)

Scrieți un program care citește un număr natural nenul n ($n < 100$) și un șir de n numere naturale nenule cu cel mult patru cifre fiecare, apoi afișează pe ecran șirul ordonat crescător în funcție de suma cifrelor corespunzătoare fiecărui termen al său. Dacă două numere au aceeași sumă a cifrelor, se va afișa mai întâi cel mai mic dintre ele.

Exemplu: pentru $n=5$ și șirul (701, 1000, 44, 99, 143) se va afișa șirul (1000, 44, 143, 701, 99).

Ați înțeles?

Probleme cu răspuns scurt

►1. (Bacalaureat iulie 2008, varianta 37)

În secvența de program următoare, elementele vectorului a au valorile (1, 2, 3, 4, 5), în această ordine. Care va fi rezultatul afișat în urma executării secvenței?

```
for(i=0; i<=4; i++)
{
    s=0;
    for (j=0; j<=i; j++) s+=a[i];
    cout << s;
}
```

►2. (Bacalaureat iulie 2008, varianta 36)

Se consideră tabloul unidimensional $a = (0, 1, 3, 2)$. Indicați tabloul unidimensional b , astfel încât pentru orice i ($0 \leq i < 4$) să existe relația $a[b[i]] = b[a[i]]$.

- | | |
|-----------------|-----------------|
| a) (2, 1, 0, 3) | b) (3, 1, 0, 2) |
| c) (2, 3, 0, 1) | d) (1, 0, 2, 3) |

Fie v un vector cu $n=9$ componente numere întregi. Următoarele trei probleme se referă la algoritmul de mai jos:

```
s ← 0;
pentru (i ← 0, 1, ..., n-1) execută
    dacă (i % 2 = 1) atunci
        s ← s + v[i];
scrie s;
```

►3. (Bacalaureat iulie 2008, varianta 66)

Ce se va afișa în urma execuției secvenței dacă elementele vectorului v sunt, în ordine, (14, 3, 7, 0, -4, 3, 10, 15, 7).

►4. Transcrieți în limbajul de programare Pascal structura de selecție prezentă în cadrul algoritmului dat.

►5. Modificați o singură linie în cadrul algoritmului de mai sus, astfel încât elementele vectorului să fie parcursă în ordine inversă, iar efectul execuției algoritmului să fie același.

►6. În urma execuției secvenței de program alăturate, variabila ok , de tip întreg cu sens logic, ar trebui să aibă valoarea: 1 dacă x se găsește printre cele n elemente ale vectorului v , respectiv 0 în caz contrar. Algoritmul este corect? Justificați răspunsul, exemplificând pentru $x=8$ și vectorul $v = (6, 3, 8, 5, 1)$ cu $n=5$ elemente.

```
ok=0;
for (i=0; i<n; i++)
    if (v[i]==x)
        ok=1;
else
    ok=0;
```

Fie v un vector cu n elemente de tip întreg, unde n este un număr natural nenul. Următoarele două probleme se referă la secvența de program de mai jos:

```
i=0;
while (i<n)
{
    v[i]=i*i; i++;
}
```

►7. (Bacalaureat iulie 2008, varianta 69)

Câte atribuiri se execută în total în timpul rulării secvenței de program date.

►8. Pentru $n=4$, care vor fi elementele vectorului v după execuția secvenței?

►9. Care dintre secvențele de program de mai jos calculează corect suma primelor n elemente ale unui vector, $S = v[0] + v[1] + \dots + v[n-1]$?

- | | | |
|---------------------|-------------|------------------|
| a) | b) | c) |
| $S=0;$ | $S=0; i=0;$ | $S=0; i=0;$ |
| for (i=0; i<n; i++) | while (i<n) | do |
| $S+=v[i];$ | { | { |
| | $S+=v[i];$ | $S+=v[i];$ |
| | $i++;$ | $i++;$ |
| | } | } while (i<n-1); |

d) nici una dintre variantele anterioare

►10. (Bacalaureat iulie 2000, variantele 1 și 6)

Care dintre următoarele secvențe de instrucțiuni determină în variabila reală max cel mai mare element dintr-un șir de n numere reale $a_1, a_2, \dots, a_n$?

- | | |
|---------------------------------|-------------------------------------|
| a) $max=0;$ | b) $max=a[1];$ |
| for (i=1; i<=n; i++) | for (i=2; i<=n; i++) |
| if ($max < a[i]$) $max=a[i];$ | if ($max < a[i]$) $max=a[i];$ |
| c) $max=a[1];$ | d) $max=a[n];$ |
| for (i=2; i<=n; i++) | for (i=2; i<=n; i++) |
| if ($max > a[i]$) $max=a[i];$ | if ($max < a[i-1]$) $max=a[i-1];$ |

► 11. (Bacalaureat iulie 2008, varianta 25)

Se consideră două tablouri unidimensionale a și b , cu elementele $a=(1,3,5,9,10)$, respectiv $b=(2,4,6,7)$. Care vor fi elementele tabloului rezultat în urma interclasării tablourilor a și b .

► 12. (Bacalaureat iulie 2008, varianta 47)

Câte interschimbări trebuie efectuate în total pentru a sorta crescător prin metoda bulelor un vector cu elementele $(10,9,8,7,6,5,4,3,2,1,0)$?

► 13. Fie secvența de program
alăturată, în care toate variabilele sunt de tip întreg.

```
for (p=1,k=1;k<n;k++)
    if (v[k]==v[k-1]) p=0;
cout << p;
```

Secvența afișează 0 dacă:

- a) Toate elementele sunt distincte două câte două.
- b) Toate elementele sunt egale.
- c) Există două elemente consecutive distincte.
- d) Există numai două elemente consecutive egale.

► 14. Care dintre afirmațiile de mai jos *nu* sunt adevărate pentru secvența de program următoare?

```
p=0;
for (k=1;k<6;k++)
    if (v[k]>v[p]) p=k;
printf("%d",p);
```

- a) Ciclul `for` are patru pași.
- b) Dacă elementele vectorului $(v[0], v[1], \dots, v[5])$ sunt $5, 4, -11, 9, -12, 1$, atunci programul afișează valoarea 4.
- c) Dacă elementele vectorului $(v[0], v[1], \dots, v[5])$ sunt $3, -2, 8, 6, 11, 4$, atunci programul afișează valoarea 4.
- d) Indiferent care ar fi elementele vectorului, secvența dată nu poate afișa valoarea 0.

► 15. În secvența alăturată, v este un vector cu cinci elemente numere întregi, iar celelalte variabile sunt de tipul întreg.

```
int i,v[6],s;
for(s=5,i=1;i<6;
    s-=v[i]-v[i-1],i++);
cout << s;
```

Ultima valoare afișată va fi 0 dacă:

- a) Primul element al vectorului este 5, iar celelalte elemente sunt nule.
- b) Ultimul element al vectorului este 6, iar celelalte elemente sunt nule.
- c) Fiecare element începând cu al doilea este mai mare cu 1 decât elementul aflat înaintea lui în vector.
- d) Fiecare element începând cu al doilea este mai mic cu 1 decât elementul aflat înaintea lui în vector.
- e) Toate elementele vectorului au valoarea 5.

► 16. Se consideră vectorii u și v , cu m , respectiv, n elemente de tip întreg.

Precizați care dintre situațiile descrise mai jos *nu* sunt adevărate în urma execuției secvenței de mai jos.

```
k=0;
for(i=0;i<m;i++)
    for(j=0;j<n;j++)
        if (u[i]==v[j])
            w[k++]=u[i];
```

- a) Pentru vectorii $u=(2,-3,1)$ cu $m=3$ elemente și $v=(-3,4,2,5)$ cu $n=4$ elemente, se va obține în urma execuției secvenței vectorul $w=(2,-3)$.
- b) Pentru vectorii $u=(3,7,3)$ cu $m=3$ elemente și $v=(2,7,2,3)$ cu $n=4$ elemente, se va obține în urma execuției secvenței vectorul $w=(3,7)$.
- c) Pentru vectorii $u=(1,-2,6)$ cu $m=3$ elemente și $v=(3,4,8,9)$ cu $n=4$ elemente, la finele execuției secvenței date vectorul w nu va avea nici un element.
- d) Dacă înaintea execuției secvenței unul dintre vectorii u, v nu conține nici un element, atunci după execuția secvenței vectorul w nu va conține nici un element.



Probleme propuse

► 1. (Bacalaureat iulie 2009, varianta 82 – enunț adaptat)

Scrieți un program care primește de la tastatură un tablou unidimensional cu n elemente numere întregi, fiecare element având cel mult patru cifre, și afișează pe ecran produsul elementelor impare din tablou, sau valoarea 0 dacă nu există elemente impare.

Exemplu: pentru tabloul $v=(-3,8,5,1,2,4)$ programul va afișa numărul $-15 (-3 \cdot 5 \cdot 1)$, iar pentru tabloul $v=(12,0,4,16)$ se va tipări valoarea 0.

► 2. (Bacalaureat iulie 2009, varianta 16)

Scrieți un program care citește de la tastatură elementele unui tablou unidimensional cu exact 10 numere naturale, mai mici decât 1000, determină și afișează pe ecran, separate prin câte un spațiu, numărul de elemente din șir care sunt multipli ai numărului 13 și, în continuare, pozițiile pe care acestea se găsesc în șir. Elementele tabloului sunt numerotate de la 0 la 9.

Exemplu: dacă șirul citit este $(2, 6, 26, 14, 130, 11, 8, 23, 52, 39)$, se vor afișa numerele $4 \ 2 \ 4 \ 8 \ 9$.

Indicații: Dacă nu s-ar fi cerut să afișăm mai întâi numărul elementelor divizibile cu 13 și abia apoi pozițiile acestora, ar fi fost suficientă o singură parcurgere a vectorului, în care afișăm pozițiile elementelor cu pricina și în același timp le numărăm în variabila nx , apoi, după ieșirea din ciclu, afișăm nx . În contextul cerinței privind forma de afișare, sunt necesare două parcurgeri: cu prima parcurgere facem numărarea,

afișăm nx , iar apoi cu a doua parcurgere "vizităm" din nou toate elementele $v[i]$ (cu $i=0, 1, \dots, n-1$) și afișăm pozițiile celor divizibile cu 13.

▶ **3. (Bacalaureat iulie 2009, varianta 67 – enunț adaptat)**

Scrieți un program care primește de la tastatură un tablou unidimensional cu n elemente numere întregi, fiecare element având cel mult nouă cifre, și afișează pe ecran numărul de numere prime din tablou.

Exemplu: pentru $n=5$ și tabloul $(12, 37, 43, 6, 72)$, se va tipări valoarea 2.

▶ **4. (Bacalaureat iulie 2009 - varianta 44)**

Scrieți un program care citește de la tastatură numărul natural n ($0 < n < 100$) și un șir format din n numere întregi de cel mult 4 cifre fiecare, determină și afișează pe ecran numărul de pătrate perfecte din șir.

Exemplu: dacă $n=6$, iar șirul este format din elementele $(31, 25, 19, 11, 4, 3)$, atunci pe ecran se va afișa 2.

▶ **5. (Bacalaureat iulie 2009 - varianta 33 - enunț adaptat)**

Scrieți un program care primește de la tastatură un tablou unidimensional cu n elemente numere întregi, și afișează mesajul DA în cazul în care printre elementele tabloului x se află cel puțin un număr impar, sau afișează mesajul NU în caz contrar.

Exemplu: pentru vectorul $v=(2, 4, 4, 8, 6, 12, 12)$, programul va afișa mesajul NU.

▶ **6.** Scrieți un program care verifică dacă cele n elemente ale unui șir dat de numere întregi sunt în ordine strict crescătoare de la stânga la dreapta.

Exemplu: Elementele șirului $(2, 3, 5, 8)$ sunt în ordine strict crescătoare iar șirul $(2, 3, 8, 5)$ nu îndeplinește această cerință.

Indicații: Avem de-a face cu o testare logică. Presupunem mai întâi condiția adevărată, inițializând cu 1 o variabilă ok (de tip întreg, dar cu sens logic). Apoi, într-un ciclu dirijat de contorul i , parcurgem pozițiile elementelor începând cu al doilea, și pentru fiecare element $v[i]$ căutăm cazul contrar: dacă $v[i] \leq v[i-1]$, înseamnă că la pasul respectiv am găsit un element $v[i]$ care "strică" ordinea strict crescătoare, deci ok primește valoarea 0 (aferentă "stării" de condiție falsă). În final testăm valoarea cu care ok a ieșit din ciclu, și în funcție de aceasta afișăm un mesaj sugestiv.

▶ **7.** Fiind dat un vector v cu n elemente numere întregi, citite de la tastatură, scrieți un program care afișează toate perechile de elemente consecutive egale între ele. Fiecare astfel de pereche va fi scrisă pe un rând de ecran, între două paranteze, cu elementele sale separate prin "virgulă".

Exemplu: Pentru vectorul $v=(2, 3, 3, 8, 5, 5, 11, 11, 7, 9)$ cu $n=10$ elemente, se vor afișa perechile $(3, 3)$, $(5, 5)$ și $(11, 11)$.

▶ **8. (Bacalaureat iulie 2009, varianta 10)**

Scrieți un program care citește de la tastatură un număr natural n ($n \leq 100$) și apoi cele n elemente, numere naturale cu cel mult patru cifre fiecare, ale unui tablou unidimensional a . Programul determină și afișează pe

prima linie a ecranului suma celor n elemente ale tabloului, pe a doua linie a ecranului suma primelor $n-1$ elemente și așa mai departe, astfel încât pe linia $n-1$ se va afișa suma primelor 2 elemente, iar pe linia n primul element al tabloului. Pozițiile elementelor tabloului sunt numerotate cu $0, 1, \dots, n-1$.

Exemplu: dacă $n=4$, iar tabloul are elementele $a=(1, 2, 3, 4)$ programul va afișa valorile: 10, 6, 3, 1

Indicații: Practic avem de calculat n sume, deci vom proiecta un ciclu în care contorul p va lua pe rând valorile $1, 2, \dots, n$ și la fiecare pas calculăm și afișăm suma cu numărul de ordine p , care este de fapt suma primelor p elemente; pentru a efectua această sumă, într-un al doilea ciclu interior primului, parcurgem pozițiile i ale elementelor de la 0 la $p-1$, și la fiecare pas adăugăm elementul $v[i]$ la o variabilă s (inițializată cu 0 la fiecare pas al primului ciclu, înainte de declanșarea celui de-al doilea). Putem folosi aceeași variabilă s la fiecare pas al primului ciclu, deoarece nu se cere salvarea acestor sume, ci doar afișarea fiecăreia dintre ele, imediat ce a fost calculată.

▶ **9.** Se citește de la tastatură un șir de n numere întregi. Să se afișeze toate perechile de elemente consecutive cu proprietatea că al doilea element al perechii este egal cu suma cifrelor primului.

Exemplu: Pentru șirul $(124, 7, 12, 44, 8, 9)$ se afișează perechile $(124, 7)$, $(44, 4)$.

▶ **10. (Bacalaureat iulie 2009, varianta 28)**

Scrieți un program care citește de la tastatură un număr natural n din intervalul $[2, 10000]$ și apoi n numere reale și afișează pe ecran câte dintre cele n numere reale sunt egale cu media aritmetică a celorlalte $n-1$ numere reale.

Exemplu: pentru șirul $(7, 2, 3, 4)$ se va afișa valoarea 1 (pentru că un singur element, și anume 4, este media aritmetică a celorlalte), iar pentru șirul $(1, 1, 1, 1, 1)$ se va afișa valoarea 5.

▶ **11. (Bacalaureat iulie 2009, varianta 54)**

Scrieți un program care citește de la tastatură un număr natural nenul n ($n \leq 100$) și apoi cele n numere naturale nenule, de maximum patru cifre, reprezentând elementele unui tablou unidimensional v (cu indici de la 0 la $n-1$) și afișează câte dintre elementele v_i ($2 \leq i \leq n-1$) sunt egale cu suma celor două elemente vecine. În cazul în care nu există niciun astfel de element în tabloul v , se va afișa valoarea 0.

Exemplu: dacă $n=7$ și tabloul unidimensional v are conținutul $(10, 25, 15, 45, 30, 2, 1)$, atunci se va afișa valoarea 2 (deoarece $25=10+15$, $45=15+30$).

▶ **12.** Pentru un șir dat de n elemente numere reale, să se afișeze toate tripletele de elemente consecutive cu proprietatea că al doilea element al tripletului este media aritmetică dintre primul și al treilea element.

Exemplu: Pentru șirul $v=(2, 3, 5, 10, 15, 20, 9, -2)$ cu $n=8$ elemente, se vor afișa tripletele $(5, 10, 15)$, $(10, 15, 20)$ și $(20, 9, -2)$.

Indicații: Dacă memorăm șirul într-un vector $v = (v[0], v[1], \dots, v[n-1])$ cu n elemente, atunci tripletele de elemente consecutive sunt de forma $(v[i], v[i+1], v[i+2])$, cu $i=0, 1, \dots, n-3$. În cadrul unui ciclu, pentru fiecare astfel de triplet, testăm dacă $v[i+1] == (v[i] + v[i+2]) / 2$, iar în caz afirmativ afișăm tripletul.

► 13. Se citește de la tastatură un șir cu n elemente numere întregi. Să se construiască un alt șir cu elementele șirului dat citite invers, de la dreapta la stânga. Să se realizeze apoi aceeași inversare memorându-se însă noul șir în același vector în care s-a citit șirul inițial (fără a folosi vreun vector suplimentar).

Exemplu: Pentru șirul (1, 2, 8, 5) se va construi șirul (5, 8, 2, 1).

► 14. Se citește de la tastatură un șir de n elemente cu numere întregi. Scrieți un program care șterge elementul minim din șir, apoi afișează șirul rămas.

Exemplu: pentru șirul (7, 5, 2, 14, 8, 11) cu $n=6$ elemente, se va șterge minimul 2, rezultând astfel șirul (7, 5, 14, 8, 11).

Indicații: Memorăm șirul într-un vector $v = (v[0], v[1], \dots, v[n-1])$. Mai întâi determinăm elementul minim $\min$, precum și poziția p a acestuia, cu algoritmul explicat pe larg în acest capitol. Pentru a șterge elementul de pe poziția p , vom muta cu o poziție mai la stânga toate elementele aflate după el, adică cele de pe pozițiile $p+1, \dots, n-1$. Pentru aceasta, vom folosi un ciclu cu $i=p+1, \dots, n-1$, în care la fiecare pas elementul $v[i]$ trece pe poziția anterioară $i-1$, prin atribuirea $v[i-1] = v[i]$.

► 15. Se citește de la tastatură un șir de n elemente numere întregi. Să se insereze la mijlocul șirului media aritmetică a elementelor sale.

Exemplu: Dacă de la tastatură se citește șirul (8, -3, 12, 4) cu $n=4$ elemente, atunci noul șir obținut în urma inserării mediei 5 pe poziția din mijloc va fi (8, -3, 5, 25, 12, 4).

► 16. (Bacalaureat iulie 2009 - varianta 24 - enunț adaptat)

Se citesc de la tastatură o valoare naturală nenulă n , ($3 \leq n \leq 100$), apoi cele n elemente, distincte, ale unui tablou unidimensional x . Fiecare dintre aceste elemente este un număr natural având cel mult patru cifre. Folosind un algoritm eficient, realizați un program care va calcula și va afișa pe ecran media aritmetică a elementelor care ar rămâne în tabloul x dacă s-ar elimina valoarea minimă și valoarea maximă din tablou. Valoarea afișată va avea cel mult 3 cifre după virgulă.

Exemplu: dacă se citesc pentru n valoarea 5, iar pentru tabloul x valorile (1, 9, 4, 8, 5), programul va afișa una dintre valorile 5.667 sau 5.666.

► 17. Se citesc de la tastatură cele n elemente ale unui șir dat de numere întregi. Să se afișeze toate perechile de elemente ale șirului (nu neapărat consecutive) cu proprietatea că ambele elemente ale perechii au aceeași sumă a cifrelor.

Exemplu: În șirul (14, 129, 221, 65, 409, 5, 32) cu $n=7$ elemente, avem perechile (14, 221), (14, 5), (14, 32), (221, 5), (221, 32) și (5, 32) cu suma cifrelor 5.

► 18. Se dă un vector v cu n elemente numere întregi. Să se copieze într-un alt vector u elementele strict pozitive ale vectorului inițial.

Exemplu: Pentru $n=6$ și vectorul $v=(7, -8, -15, 2, 0, 13)$, vom obține vectorul $u=(7, 2, 13)$.

Indicații: Algoritmul este foarte asemănător cu cel din problema rezolvată R.V.9. Elementele strict pozitive din vectorul v vor fi adăugate pe rând la sfârșitul vectorului u (inițial gol). Folosim un contor k , în care vom păstra în permanență poziția (indicele "căsuței") la care am ajuns cu adăugarea în vectorul u . Evident, inițial $k=0$, pentru că începem adăugarea cu prima poziție. Parcurgem într-un ciclu pozițiile $i=0, 1, \dots, n-1$ ale elementelor vectorului dat v , și la fiecare pas, dacă elementul $v[i]$ este strict pozitiv, atunci:

- îl adăugăm la sfârșitul vectorului u , pe poziția dată de valoarea curentă a lui k ($u[k] = v[i]$);

- incrementăm cu 1 variabila k , pentru a pregăti "căsuța" din vectorul u în care se va memora următorul număr strict pozitiv din vectorul v (la un pas următor al ciclului).

În final, numărul de elemente ale vectorului u este chiar k .

► 19. Fiind dat un vector v cu n elemente numere întregi, să se construiască alți doi vectori: primul va conține numai elementele pare, iar al doilea numai elementele impare ale vectorului inițial.

Exemplu: Pentru $n=7$ și $v=(8, -6, 11, 14, 10, -5, -214)$, vor rezulta vectorii $x=(8, -6, 14, 10, -214)$ și $y=(11, -5)$.

► 20. (Bacalaureat iulie 2009, varianta 93)

Scrieți un program care citește de la tastatură un număr natural n ($1 \leq n \leq 99$), impar, și construiește în memorie un tablou unidimensional $A=(A_1, A_2, \dots, A_n)$ cu elementele mulțimii $\{1, 2, \dots, n\}$ astfel încât elementele de pe poziții impare formează șirul crescător $1, 2, \dots, [(n+1)/2]$, iar elementele de pe poziții pare șirul descrescător $n, n-1, \dots, [(n+1)/2]+1$.

Exemplu: pentru $n=11$ se va construi tabloul $A=(1, 11, 2, 10, 3, 9, 4, 8, 5, 7, 6)$.

► 21. (Bacalaureat iulie 2008, varianta 93)

Să se introducă într-un vector, apoi să se afișeze ca atare, următorul șir de numere: 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, ...

Numărul de elemente câte vor fi introduse se va preciza anticipat de la tastatură.

Indicații: Putem observa următoarea grupare a elementelor șirului: (1), (1, 2), (1, 2, 3), (1, 2, 3, 4), etc... Grupa 1 conține valorile de la 1 la 1, grupa 2 valorile de la 1 la 2, ș.a.m.d., deci o grupă oarecare i va conține valorile de la 1 la i , pe care le afișăm banal într-un ciclu. Numai că, neștiindu-se numărul grupelor ci numărul elementelor, pentru trecerea grupelor în revistă nu se poate folosi tot un ciclu `for`, ci un ciclu `while`, în care să se numere elementele afișate, și din care să se iese forțat atunci când s-au parcurs n valori. În rest, adăugarea elementelor pe rând într-un vector nou se încadrează în algoritmul clasic.

► 22. Se dau două mulțimi definite prin intermediul vectorilor u și v , cu m respectiv n elemente. Să se memoreze în vectorul w și apoi să se afișeze reuniunea celor două mulțimi (elementele comune și necomune luate o singură dată). La citirea fiecăruia dintre cei doi vectori se va asigura îndeplinirea condiției de mulțime: elementele să fie distincte și în ordine crescătoare.

Exemplu: Pentru mulțimile $u=(2,3,8,9)$ și $v=(3,8,11)$, va rezulta vectorul-reuniune $(2,3,8,9,11)$.

► 23. (Bacalaureat iulie 2009, varianta 31 - enunț adaptat)

Se citește de la tastatură un tablou unidimensional x cu cel mult 100 de elemente, numere naturale cu cel mult 4 cifre fiecare, și un număr natural n ($n \leq 100$), ce reprezintă numărul efectiv de elemente ale tabloului x . Scrieți un program care va afișa tabloul obținut în urma schimbării poziției doar a elementelor impare din tablou astfel încât acestea să apară în ordinea crescătoare a valorilor lor.

Exemplu: pentru $n=6$ și $x=(7,11,2,-8,-3,10)$ programul va afișa tabloul $(-3,7,2,-8,11,10)$.

Indicații: Practic algoritmul de sortare prin metoda interschimbărilor directe trebuie modificat, astfel încât la fiecare comparare să se compare între ele doar două elemente $x[i]$ și $x[j]$ care sunt ambele impare. În cazul în care vreți să folosiți "metoda bulilor", modificările sunt mai profunde!

► 24. (Bacalaureat iulie 2009, varianta 41 - enunț adaptat)

Se citește de la tastatură un număr natural n și un tablou unidimensional a cu n elemente numere întregi. Scrieți un program care afișează valoarea 1 dacă toate elementele tabloului a sunt distincte și dacă diferența absolută a oricăror două elemente vecine din tablou este diferită de 1, respectiv 0 în caz contrar.

Exemplu: pentru tabloul $(7,2,8,11,13)$ se va afișa valoarea 1, iar pentru tablourile $(5,3,9,5)$ și $(2,-3,7,6)$ se va afișa valoarea 0.

► 25. (Bacalaureat iulie 2009, varianta 52)

Scrieți un program care citește de la tastatură un număr natural nenul n ($n \leq 100$) și apoi n numere naturale, de maximum patru cifre fiecare, reprezentând elementele unui tablou unidimensional. Programul afișează mesajul DA în cazul în care elementele tabloului pot fi rearanjate astfel încât să formeze o progresie aritmetică, iar în caz contrar afișează mesajul NU.

Exemplu: dacă $n=6$ și tabloul unidimensional are conținutul alăturat, atunci se va afișa DA.

5 10 30 15 25 20

► 26. (Bacalaureat iulie 2008, varianta 24)

Scrieți un program care citește de la tastatură un număr natural nenul n ($n \leq 1000$), apoi construiește în memorie și afișează pe ecran un tablou unidimensional a , având n elemente, ale cărui capete vor fi completate cu toate numerele din mulțimea $\{1,2,\dots,n\}$ luate alternativ, astfel: valoarea 1 pe prima

poziție, valoarea 2 pe ultima poziție, valoarea 3 pe a doua poziție, valoarea 4 pe penultima poziție, ș.a.m.d.. Elementele tabloului creat se vor afișa cu câte un spațiu între ele.

Exemplu: pentru $n=9$, tabloul va fi $(1,3,5,7,9,8,6,4,2)$.

► 27. Se citește de la tastatură un șir de n elemente numere întregi. Să se afișeze elementele șirului în "pagini" de câte m rânduri, fiecare rând conținând câte m elemente separate prin spații sau virgule (cu excepția ultimei pagini care poate conține mai puțin de m rânduri, respectiv a ultimului rând al "ultimei pagini" care poate conține mai puțin de m elemente). După afișarea fiecărei "pagini" programul va aștepta apăsarea unei taste pentru confirmarea trecerii la "pagina" următoare.

Exemplu: Pentru $m=3$, $n=22$ și șirul $(8,2,4,56,87,91,12,41,62,44,65,21,87,30,2,10,35,99,70,13,18,24)$ ecranul va arăta astfel:

8,2,4
56,87,91
12,41,62

44,65,21
87,30,2
10,35,99

70,13,18
24

► 28. (Bacalaureat iulie 2009, varianta 55)

Scrieți un program care citește de la tastatură un număr natural nenul n ($n \leq 100$) și $2 \cdot n$ numere naturale de maximum trei cifre; primele n valori reprezintă elementele tabloului unidimensional a , iar următoarele n elementele tabloului unidimensional b ; fiecare tablou are elementele numerotate începând de la 0. Programul construiește în memorie și afișează pe ecran cele n elemente ale unui tablou unidimensional c , în care orice element c_i ($0 \leq i \leq n$) se obține conform definiției următoare:

$$c_i = \begin{cases} a_i \text{ concatenat cu } b_i, & \text{dacă } a_i < b_i \\ b_i \text{ concatenat cu } a_i, & \text{altfel} \end{cases}$$

Exemplu: dacă $n=3$ și tablourile a și b au conținutul $a=(12,123,345)$ și $b=(1,234,15)$, atunci conținutul tabloului c este următorul: $(112,123234,15345)$.

► 29. Fiind dat un vector v cu n elemente numere întregi, să se construiască un alt vector u care să conțină numai pătratele perfecte din v , în ordine crescătoare. Aceste pătrate perfecte vor fi copiate de la început în ordine crescătoare, fără a se aplica vectorului u nici un algoritm de sortare.

Exemplu: Pentru vectorul $v=(34,9,45,64,18,16,39)$, se va crea vectorul $u=(9,16,64)$.

Indicații: Atenție, algoritmul prezentat ca indicații la problema 18 nu ne este de nici un folos în acest caz. Presupunând că la un pas oarecare i al ciclului de parcurgere a vectorului v (cu $i=0,1,\dots,n-1$) avem în u adăugate deja k elemente $u[0], u[1], \dots, u[k-1]$ ordonate crescător, adăugarea elementului $v[i]$ nu se va face la sfârșitul lui u ! Va trebui să găsim poziția pe care ar trebui să o ocupe acea valoare $v[i]$ în vectorul u , astfel încât, după introducerea ei, vectorul u să rămână ordonat crescător. Pentru aceasta, căutăm în u un indice j , astfel încât $u[j] \leq v[i] \leq u[j+1]$, și, după ce l-am găsit, inserăm pe $v[i]$ pe poziția j în vectorul u .

► **30. (Bacalaureat iulie 2009, varianta 12)**

Se citesc de la tastatură două valori naturale m și n ($m \leq 100$, $n \leq 100$), iar apoi $m+n$ numere întregi de cel mult nouă cifre fiecare. Dintre cele $m+n$ numere citite, primele m sunt ordonate strict crescător, iar următoarele n sunt, de asemenea, ordonate strict crescător. Se cere să se afișeze pe ecran câte din cele $m+n$ numere au fost citite o singură dată.

Exemplu: pentru $m=6$ și $n=9$ și valorile 1, 2, 3, 4, 7, 20, 3, 5, 7, 8, 9, 10, 12, 20, 24, se va afișa valoarea 9 (numerele care au fost citite o singură dată sunt 1 2 4 5 8 9 10 12 24).

a) Descrieți un algoritm de rezolvare a acestei probleme, eficient din punct de vedere al timpului de executare și al spațiului de memorie utilizat, explicând în ce constă eficiența acestuia.

b) Scrieți programul corespunzător algoritmului descris.

► **31. (Bacalaureat iulie 2008, varianta 91)**

Pentru un număr natural n citit de la tastatură ($0 < n < 100$), se cere să se construiască un vector cu n componente numere naturale din mulțimea $\{0, 1, 2\}$, astfel încât să nu existe două elemente alăturate egale, iar suma oricăror trei elemente consecutive să fie egală cu 3. Scrieți programul care generează în memorie vectorul, apoi scrie componentele acestuia pe ecran, în ordine, cu spații între ele.

Exemplu: pentru $n=4$, se vor afișa valorile 1 2 0 1.

Indicații: Problema pare grea, dar cu o idee genială ea devine banală. Practic, dacă multiplicăm secvența (1, 2, 0) de $\lfloor n/3 \rfloor$ ori, obținem un șir care respectă cele două cerințe. Iată un exemplu: pentru $n=11$ avem șirul (1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0). Observăm că sumele a trei elemente alăturate sunt fie $1+2+0$, fie $2+0+1$, fie $0+1+2$, deci întotdeauna 3.

► **32. (Bacalaureat iulie 2009, varianta 66 - enunț adaptat)**

Se citesc de la tastatură un număr natural nenul n ($1 \leq n \leq 10000$) precum și un tablou unidimensional a care conține n valori întregi, fiecare dintre aceste valori având cel mult nouă cifre. Realizați un program care determină și afișează pe ecran cel mai mare divizor comun al elementelor tabloului a .

Exemplu: în urma apelului, pentru $n=5$ și tabloul unidimensional (12, 36, 48, 6, 60), se va returna 6.

► **33.** La ora de educație fizică, în vederea apelului de prezență, elevii unei clase s-au așezat într-un șir după înălțime, în ordine crescătoare de la stânga la dreapta. Profesoara dorește să inverseze ordinea elevilor în șir, astfel încât ei să fie aranjați tot după înălțime, dar în ordine descrescătoare de la stânga la dreapta. Simulați într-un program algoritmul pe care-l aplică profesoara. De la tastatură se citesc numărul de elevi n , precum și înălțimile elevilor. Programul va trebui să testeze dacă așezarea inițială a elevilor respectă condiția de ordine crescătoare, apoi să afișeze elevii după re-ordonarea dictată de profesoară.

Indicații: Se interschimbă între ei primul elev din șir cu ultimul, al doilea cu penultimul, al treilea cu antepenultimul, ș.a.m.d. Dacă numărul elevilor este impar, atunci elevul aflat în mijlocul șirului va rămâne pe locul inițial.

► **34. (Bacalaureat iulie 2009, varianta 34)**

Scrieți programul care citește de la tastatură un număr natural n ($n < 100$), apoi n numere naturale de cel mult trei cifre fiecare și afișează pe ecran cel mai mare număr de valori pare (dintre cele n citite) care s-au citit consecutiv de la tastatură.

Exemplu: pentru $n=8$ și numerele 12, 7, 4, 16, 10, 3, 6, 6 se va afișa 3.

► **35.** Se citesc de la tastatură n fracții de forma $(a_1/b_1, a_2/b_2, \dots, a_n/b_n)$. Să se afișeze fracțiile simplificate "la maximum", precum și fracția ce reprezintă suma celor n fracții. Reamintim: o fracție se simplifică "la maximum" împărțind numărătorul și numitorul la cel mai mare divizor comun al lor.

Exemplu: Dacă cele $n=3$ fracții de intrare sunt (6/6, 12/18, 50/20), atunci se vor afișa fracțiile (1/1, 2/3, 5/2), iar suma acestora este (25/6).

Indicații: Memorăm numărătorii respectiv numitorii celor n fracții în doi vectori $(a[0], a[1], \dots, a[n-1])$ respectiv $(b[0], b[1], \dots, b[n-1])$. Într-un ciclu cu $i=0, 1, \dots, n-1$, simplificăm fiecare fracție $a[i]/b[i]$ astfel: determinăm c.m.m.d.c. $(a[i], b[i])$, apoi împărțim $a[i]$ și $b[i]$ la acest c.m.m.d.c. Pentru a însuma cele n fracții, trebuie mai întâi să determinăm cel mai mic multiplu comun al numitorilor $(b[0], b[1], \dots, b[n-1])$, care se poate calcula după relația $NC = (b[0] * b[1] * \dots * b[n-1]) / \text{c.m.m.d.c.}(b[0], b[1], \dots, b[n-1])$. Calculul celui mai mare divizor comun al numitorilor $(b[0], b[1], \dots, b[n-1])$ folosește proprietatea de tranzitivitate: astfel, într-un ciclu, vom determina succesiv $d1 = \text{c.m.m.d.c.}(b[0], b[1])$, $d2 = \text{c.m.m.d.c.}(d1, b[2])$, ș.a.m.d. În final, într-un alt ciclu se determină numărătorul fracției sumă: $a[0]$ se înmulțește cu $NC/b[0]$, $a[1]$ cu $NC/b[1]$, etc., adunându-se aceste rezultate.

► **36. (Bacalaureat iulie 2008, varianta 45 - enunț adaptat)**

Se citește de la tastatură un șir de numere naturale mai mici sau egale decât 10000. Șirul are cel mult 100 de valori și se termină atunci când se introduce o valoare negativă (care nu face parte din el). Scrieți un program care afișează pe ecran toate numerele impare din șir, în ordine crescătoare, separate prin câte un spațiu. În cazul în care un anumit element se repetă în șir, el se va afișa o singură dată.



Capitolul VI: Fișiere



VI.1. Fișiere text



VI.1.1 Noțiunea de fișier de date

Noțiunea de fișier vă este cunoscută încă din primele ore de informatică. Atunci ați învățat să realizați un desen sau să scrieți un text, pe care l-ați salvat, unde credeți, într-un fișier ! Fiecare program C++, "butonat" caracter cu caracter de la tastatură, este de fapt tot un fișier, numit *sursa programului*. Tot la începutul anului, ați aflat că un sistem de operare este un ansamblu de programe, adică o colecție de fișiere, având rolul de a asigura exploatarea propriu-zisă a calculatorului. În concluzie, putem spune că ați făcut cunoștință până acum cu o mare varietate de tipuri de fișiere.

Ați învățat tot în primele capitole că fișierele sunt memorate pe suporturi externe, numite *discuri* (*hard-disk*, *floppy-disk*, *CD* etc.), care le păstrează în permanență (adică spunem că aceste suporturi sunt nevolatile).

Orice fișier se caracterizează printr-un nume sub care este identificat de către sistemul de operare, nume alcătuit din două părți: numele propriu-zis dat de utilizator și extensia care indică tipul fișierului, separate prin caracterul "*punct*". De exemplu: *ecuatii.pas*, *cerere.doc*, *image.bmp* etc.

Numim fișier, o colecție de informații care satisfac un anumit scop, memorate pe un suport extern, și accesată printr-un nume (numele fișierului).

În continuare vom vedea că și programele C++ pot lucra cu fișiere. Până acum am învățat că datele cu care lucrează un program (constante, variabile, tablouri, mulțimi, structuri etc.) pot fi citite de la tastatură. În timpul execuției unui program, aceste date sunt stocate în memoria internă RAM. Pentru că memoria RAM este volatilă, ele se pierd odată cu încheierea execuției programului, așa încât dacă vrem să mai executăm odată programul, trebuie să le introducem din nou.

Acest fapt poate constitui o problemă serioasă, deoarece programele concrete "din viața reală" folosesc de obicei un volum mare de date. Gândiți-vă numai la o bibliotecă publică unde sunt câteva sute sau poate chiar mii de cărți. Am învățat în capitolul "*Înregistrări*", unde am mai dat acest exemplu, că informațiile despre cărțile din bibliotecă (titlul, autorul etc.) pot fi memorate într-o înregistrare și pot fi citite de la tastatură. Dar ce face bibliotecara în condițiile în care în momentul ieșirii din program toate datele se pierd ? Le introduce din nou

de fiecare dată când vrea să ruleze programul, spre exemplu fie și numai pentru a căuta o anumită carte ? Ar fi absurd ! Nu există decât o singură soluție: datele respective să fie memorate într-un fișier, pe un disc, unde vor fi păstrate în permanență. Ajungem astfel la concluzia că și datele cărților din bibliotecă pot fi păstrate într-un așa-numit **fișier de date**.

Sistemul de operare "vede" fișierele de date la fel ca pe orice alte fișiere (recunoscându-le, reamintim, prin numele lor), motiv pentru care și asupra acestora se pot aplica operațiile care se pot realiza cu orice fel de fișiere sub controlul sistemului de operare: copiere, mutare, ștergere, redenumire etc.

În plus însă, un fișier de date poate fi scris sau citit de către utilizator. Astfel, revenind la exemplul cu biblioteca:

- ✓ informațiile despre cărți vor fi scrise o singură dată într-un fișier de date, unde vor fi păstrate în permanență pe disc;
- ✓ de fiecare dată când este rulat, în loc să citească datele de la tastatură, programul le va citi din respectivul fișier.

Categoric în acest moment aveți o nelămurire: cum poate fi scris un fișier de date ? Răspunsul este foarte simplu: fie manual (prin intermediul unui editor de texte, de exemplu poate fi folosit chiar editorul limbajului Pascal în care vă scrieți sursele programelor), fie prin intermediul unui alt program.

Despre modul cum un program poate să scrie într-un fișier de date sau poate să citească dintr-un astfel de fișier, vom învăța în lecțiile următoare.



VI.1.2. Clasificarea fișierelor de date

Din punctul de vedere al conținutului, fișierele de date în Pascal se împart în trei categorii:

- **fișiere cu tip:** componentele sunt toate de același tip (predefinit, sau definit de utilizator);
- **fișiere text:** datele sunt memorate ca o succesiune de caractere dispuse pe unul sau mai multe rânduri;
- **fișiere fără tip:** informațiile sunt stocate sub forma unor blocuri de octeți de dimensiune fixă.

Din punctul de vedere al accesului la componente, fișierele de date sunt de două tipuri:

- **cu acces secvențial:** pentru a accesa o anumită componentă trebuie parcurse toate componentele dinaintea acesteia;
- **cu acces direct:** orice componentă poate fi accesată imediat prin intermediul numărului ei de ordine în fișier.



VI.1.3. Noțiunea de fișier text

Un fișier text conține una sau mai multe *linii de caractere de lungime variabilă*. O astfel de linie se mai numește *rând în fișier*. Fiecare rând, mai puțin ultimul, se încheie printr-un "marcaj de sfârșit de linie" alcătuit din caracterele CR ("Carriage Return") și LF ("Line Feed"). Sfârșitul fișierului este marcat prin caracterul EOF ("End Of File"). *

Caracterele dintr-un fișier text pot fi atât caractere *tipăribile* (litere mari și mici, cifre, caractere speciale), cât și caractere *netipăribile*. Acestea din urmă se mai numesc și *caractere albe* (ENTER, LF, SPACE, BACKSPACE, TAB etc.).

Într-un fișier text, din cauză că rândurile de caractere au lungimi diferite, nu putem identifica poziția unui anumit caracter față de începutul fișierului. De aceea, accesul într-un fișier text se face secvențial: pentru a ajunge la un caracter oarecare din interiorul fișierului trebuie să "trecem" peste toate caracterele aflate înaintea acelui caracter în fișier.

Pentru a putea prelucra datele dintr-un fișier text acesta trebuie deschis, iar după încheierea prelucrării fișierul trebuie închis la loc.

Orice fișier text se caracterizează printr-o variabilă specială numită *indicator de fișier*. Acest indicator "arată" în orice moment primul caracter din fișier ce poate fi prelucrat.



VI.1.4. Noțiunea de descriptor de fișier

Așa cum am mai spus, orice fișier, inclusiv un fișier text, se caracterizează printr-un nume sub care este identificat de către sistemul de operare (alcătuit din două părți: numele propriu-zis dat de utilizator, și extensia care indică tipul fișierului, separate prin caracterul "punct"). De obicei, pentru fișierele text se folosește extensia `txt` (de la "text") sau `dat` (de la "fișier de date"). De exemplu: `numere.txt`, `ecuatii.dat` etc.

Pe de altă parte, un program C++ nu este capabil să recunoască numele sub care este cunoscut un fișier de către sistemul de operare, motiv pentru care avem nevoie de o variabilă care să identifice fișierul. O astfel de variabilă trebuie declarată, la fel ca orice variabilă, în secțiunea de declarații a programului. Variabilele care desemnează fișierele se numesc **descriptori de fișier**, și aparțin unui tip de date predefinit, numit "*tipul pointer către fișier*". Astfel, dacă avem două fișiere și notăm cu `f` și `g` descriptorii atașați lor, atunci declarațiile descriptorilor ca variabile arată astfel:

Exemplu: `FILE *f,*g;`

Pentru a vă face să înțelegeți ce înseamnă tipul de date "*pointer către tipul fișier*" și ce semnificație are caracterul "*" plasat în fața descriptorilor `f` și `g`, ar trebui să vă prezentăm toată teoria pointerilor în C++. Aceasta însă depășește cu mult cadrul materiei de clasa a IX-a, motiv pentru care singura soluție este, cel puțin deocamdată, să luați lucrurile ca atare. În clasa a XI-a, când veți studia capitolul "*Pointeri*", veți înțelege în profunzime sensul declarațiilor de variabile de genul celei de mai sus, dar până atunci va trebui să le folosiți pur și simplu, fără să vă mai bateți capul cu interpretarea lor.

Pentru a putea folosi efectiv un fișier text într-un program, trebuie făcută o legătură între descriptorul de fișier (*fișierul logic*) și numele sub care este recunoscut de către sistemul de operare (*fișierul fizic*). Această legătură se numește **asignarea fișierului**.



VI.1.5. Prelucrarea fișierelor text cu instrumente care folosesc obiecte

În C++, prelucrarea fișierelor text se poate face și cu ajutorul unor instrumente mai "evolute", care utilizează așa-numita "programare pe obiecte". Întrucât, la fel ca pointerii, și capitolul "*Obiecte*" se studiază deabia în clasa a XI-a, ne este aproape imposibil să explicăm pe înțelesul vostru mecanismul de funcționare a instrucțiunilor care tratează fișierele din punct de vedere obiectual. De aceea, din nou suntem nevoiți, așa cum am făcut când am prezentat declararea descriptorilor de fișier ca variabile, să vă rugăm să luați ca atare anumite simboluri pe care le veți întâlni, fără a încerca să înțelegeți semnificația lor. Pentru a vă face, cât de cât, o imagine asupra a ceea ce înseamnă programarea pe obiecte, o să încercăm o definiție rudimentară a noțiunii de "obiect", dar, așa cum am spus, fără a intra în amănunte.

Extensia "++" a limbajului C++, ne permite să definim tipuri abstracte de date, care să cuprindă, pe lângă date de diverse tipuri, și funcții pentru prelucrarea respectivelor date. Un astfel de tip se numește **clasă**. Datele și funcțiile care alcătuiesc o clasă poartă denumirea de **membri ai clasei**. Funcțiile membre ale unei clase se mai numesc și **metode**. Datele membre ale unei clase sunt de obicei variabile, simple sau structurate (vectori, matrici, șiruri, etc).

În momentul de față nu putem să vă dăm nici cel puțin un exemplu de "clasă", câtă vreme nu ați studiat nici măcar funcțiile în C++. La începutul clasei a IX-a ați învățat doar că instrucțiunile unui program C++ pot fi grupate în mai multe entități numite *funcții*, și că, pe lângă funcțiile pe care le definim noi, în cadrul unui program putem folosi niște așa-numite "funcții predefinite ale limbajului". Acestea au fost scrise de către autorii limbajului, fac parte intrinsecă din limbaj, și le putem utiliza oricând în programe, în care scop este suficient să le apelăm. Dealtfel, pe parcursul clasei a IX-a ați întâlnit numeroase programe care apelau funcții predefinite precum `sqrt`, `floor`, `abs`, etc. Ei bine, cam la fel se pune problema și în ceea ce privește clasele și obiectele. Atunci când scriem un

program putem să ne definim propriile clase, dar, pe de altă parte, limbajul C++ este prevăzut cu niște *clase predefinite*, care nu mai trebuie proiectate, fiind posibil să le folosim direct, la fel cum folosim funcțiile predefinite. O astfel de clasă predefinită este ceea ce numim "*clasa fișier*", cu numele **fstream**, concepută pentru utilizarea fișierelor text. Mai multe lucruri despre clase nu vă spunem, pentru că oricum nu ați mai înțeles nimic mai departe !

Definiția în sine a acestei clase, împreună cu toate funcțiile și metodele predefinite pentru prelucrarea fișierelor text, se găsesc în header-ul **fstream.h**. De aceea, în orice program în care dorim să folosim un fișier text tratat obiectual, pentru a putea prelucra acel fișier cu ajutorul funcțiilor predefinite dedicate acestui scop, este necesar ca la începutul programului să includem header-ul **fstream.h**.

```
#include <fstream.h>
```

În lecțiile anterioare v-am prezentat funcțiile predefinite **fopen**, **fclose**, **feof**, **fscanf** și **fprintf**, cu ajutorul cărora se poate prelucra un fișier în versiunea "C standard", fără a folosi obiectele specifice extensiei "++". A venit momentul să vă spunem care sunt instrumentele necesare pentru prelucrarea obiectuală a unui fișier, adică să descoperiți instrumentele echivalente funcțiilor de mai sus, pentru deschiderea și închiderea unui fișier, respectiv pentru citire, scriere și testarea sfârșitului de fișier.

Asignarea, deschiderea și închiderea unui fișier text

Sintaxa: `fstream <descr> (<nume>, <tip>)`

- **fstream** → este numele "clasei fișier", cu ajutorul căreia se tratează fișierele ca și obiecte, aflată, așa cum spuneam, în header-ul **fstream.h** (faptul că header-ul și clasa se numesc la fel, e o simplă coincidență);
- **<descr>** → reprezintă descriptorul atașat fișierului (care se notează de obicei cu **f**, **g**, **h**, etc.);
- **<nume>** → acest parametru desemnează numele fizic sub care este recunoscut fișierul de către sistemul de operare (alcătuit din numele propriu-zis, caracterul "punct" și extensia; la fel ca și în cazul deschiderii cu **fopen**, acesta poate să conțină și calea de acces la fișier, alcătuită dintr-un lanț de folder-e și subfolder-e, separate între ele prin "\\"; în absența unei astfel de căi, se subînțelege faptul că fișierul în cauză se află în folder-ul curent;
- **<tip>** ← este un parametru care identifică tipul operației ce urmează a fi executată asupra fișierului, și poate avea două forme, "**ios::in**" și "**ios::out**", cu următoarele semnificații:
 - dacă vrem să citim din fișier parametrul este **ios::in** ("in" provine de la "input", adică "intrare");

- dacă vrem să scriem în fișier parametrul este **ios::out** ("out" provine de la "output", adică "ieșire")

Rolul simbolului "::**" este imposibil de explicat la nivelul clasei a IX-a, fiind vorba despre referirea unui obiect ce aparține unei clase, aspect care, precum spurtem mai înainte, va fi studiat deabia în clasa a XI-a.**

Prin această instrucțiune s-a realizat automat atât asignarea descriptorului pentru fișierul respectiv, cât și deschiderea lui, în vederea citirii din el sau scrierii în el.

Exemplu:

```
♥ fstream f("numere.txt", ios::in);
– deschide pentru citire fișierul numere.txt, cu descriptorul f, aflat în folder-ul curent;
♥ fstream g("numere2.txt", ios::out);
– deschide pentru scriere fișierul numere2.txt, cu descriptorul g, aflat în folder-ul curent;
♥ fstream f("D:\\Fisiere\\numere.txt", ios::in);
– deschide pentru citire fișierul numere.txt, cu descriptorul f, aflat pe partiția "D" în folder-ul "Fisiere";
♥ fstream g("D:\\Fisiere\\numere2.txt", ios::out);
– deschide pentru scriere fișierul numere2.txt, cu descriptorul g, aflat pe partiția "D" în folder-ul "Fisiere";
```

După ce am terminat de prelucrat un fișier text, el trebuie închis. Instrucțiunea de închidere este următoarea:

Sintaxa: `<descr>.close()`

- **<descr>** → reprezintă descriptorul atașat fișierului;
- **close()** → este funcția predefinită, fără parametri, care realizează efectiv închiderea;

Exemplu:

```
f.close();
Închide fișierul cu descriptorul f;
```

După cum ați observat, închiderea unui fișier se realizează la fel, indiferent dacă a fost deschis pentru citire sau pentru scriere. Prezența caracterului "punct" între descriptor și identificatorul **close** iarăși nu poate fi explicată momentan. Ea ține tot de teoria obiectelor, mai precis de modul în care este referită funcția **close** ca membră a "clasei fișier" **fstream**.

Citirea dintr-un fișier text

Citirea dintr-un fișier text tratat obiectual este similară cu citirea de la tastatură, cu deosebirea că cuvântul cheie **cin** se înlocuiește cu descriptorul

fișierului.

Sintaxă:

`<descr> >> var1 >> var2 >> varn`

La un apel al instrucțiunii, putem citi deodată mai multe variabile, iar pentru aceasta este suficient să enumerăm identificatorii `var1, var2, ..., varn` ai acestora, punând în fața fiecăruia simbolul ">>".

Din nou ne este imposibil să vă facem să înțelegeți în profunzime de unde și până unde această înlocuire a lui `cout` cu descriptorul fișierului. Pentru a vă face o imagine vagă asupra mecanismului în cauză, trebuie să vă spunem că, din punct de vedere al programării pe obiecte, tastatura este privită tot ca un fișier, dar unul mai special, numit fișierul standard de intrare, de unde se preiau date de intrare, iar cuvântul cheie `cin` nu este altceva decât descriptorul acestui fișier special. Cu alte cuvinte, în loc să citim din fișierul standard de intrare cu descriptorul `cin`, vrem să citim dintr-un fișier text "obișnuit", identificat printr-un descriptor, și de aceea înlocuim cuvântul `cin` cu identificatorul descriptorului de fișier.

În timpul citirii, compilatorul sare automat peste toate caracterele albe, adică peste tot ceea ce înseamnă spații, tab-uri și marcaje de sfârșit de rând. Acesta este un mare avantaj, spre deosebire de citirea cu `fscanf`, unde trebuia să avem evidența tuturor caracterelor albe, și să le citim ca atare. De asemenea, la citirea cu `fscanf` trebuia să știm exact pentru fiecare valoare citită câte caractere și eventual câte zecimale are, pentru a include aceste informații în specificatorul de format, lucru care aici nu e cazul.

Exemplu:

Fie fișierul `date.in` cu descriptorul `f`, având conținutul din imaginea de mai jos, precum și secvența de program următoare:

```
long a,b,c;
float x;
fstream f("C:\\Fisiere\\date.in",ios::out);
f >> a >> b >> c >> x;
f.close();
```

7 -11 3
9.25

Printr-o singură instrucțiune, trebuie citite din fișier patru variabile, și anume `a`, `b`, `c` și `x`. Ca să reușească citirea, în fișier trebuie să existe patru valori, ale căror tipuri trebuie să fie compatibile cu tipul variabilelor în care vrem să le citim. Dacă ne uităm cu atenție, observăm potrivirea: în fișier avem trei valori întregi și una reală, în această ordine, iar dintre variabile, primele trei sunt de tipul `int` și a patra de tipul `float`. Prin urmare, în urma citirii, variabilele vor primi valorile `a=7`, `b=-11`, `c=3` și `x=9.25`. Faptul că primele trei valori sunt pe un rând iar a patra valoare pe al doilea rând nu afectează citirea, întrucât marcajul de sfârșit de rând va fi "sărit" automat, la fel ca și spațiile dintre numere.

Scrierea într-un fișier text

Scrierea într-un fișier text tratat obiectual este foarte asemănătoare cu afișarea pe ecran, singura deosebire constând în faptul că cuvântul cheie `cout` se înlocuiește cu descriptorul fișierului. Este vorba aici despre o similitudine de genul

celei întâlnite în cazul citirii: atunci când afișăm ceva pe monitor, ecranul acestuia este considerat fișierul standard de ieșire cu descriptorul `cout`.

Sintaxă:

`<descr> << val1 << val2 << ... << valn`

La un apel al instrucțiunii putem scrie deodată mai multe valori, iar simbolurile `val1, val2, ..., valn` identifică aceste valori. Ele pot fi, la fel ca și în cazul scrierii pe ecran, *constante numerice*, *caractere*, *texte* (care de fapt sunt constante șir de caractere), *valori ale unor variabile*, și *valori ale unor expresii*.

Exemplu:

```
long a=2,b=5;
float x=3.14;
fstream f("date.out",ios::out);
f << a << " $$ " << b << "\n" << x << " " << (a+b)/2.;
f.close();
```

Am deschis fișierul `date.out`, cu descriptorul `f`, pentru a scrie în el (fapt consfințit de parametrul `ios::out`). Avem variabilele `a` și `b` de tip întreg cu valorile 2 respectiv 5, și variabila `x` de tipul `float` cu valoarea 3.14. Instrucțiunea de scriere în fișierul `f` este

```
f << a << " $$ " << b << "\n" << x << " " << (a+b)/2.;
```

Instrucțiunea va începe să scrie pe primul rând al fișierului următoarele elemente:

- valoarea variabilei `a`, adică 2;

- șirul de caractere " \$\$ ";

- valoarea variabilei `b`, adică 5;

- marcajul de salt la rând nou "\n";

Urmare a comenzii de scriere a acestui caracter netipăribil, se va trece la al doilea rând în fișier, unde se vor tipări una după alta;

- valoarea variabilei `x`, adică 3.14;

- caracterul "spațiu";

- valoarea expresiei "`(a+b)/2.`", adică 3.5 (calculată ca împărțire reală, grație operatorului "punc" atașată constantei întregi 2).

În concluzie, fișierul va arăta astfel după execuția instrucțiunii:

2 \$ 5
3.14 3.5

Testarea sfârșitului de fișier

Se realizează cu ajutorul funcției `eof()`, fără parametri, a cărei sintaxă este foarte asemănătoare cu cea a funcției `close()`.

Sintaxă:

`<descr>.eof()`

Funcția returnează 1 dacă s-a atins sfârșitul de fișier, respectiv 0 în caz contrar.

Exemplu:

Presupunem că avem un fișier cu descriptorul *f*, în care se găsește un șir de numere separate prin spații, și vrem să citim acest șir și să-l afișăm pe ecran. Pentru aceasta, vom citi numerele din fișier pe rând, în aceeași variabilă *x*, într-un ciclu care se va executa atâtea timp cât nu s-a atins sfârșitul de fișier, adică atâta timp cât funcția *f.eof()* nu returnează 1. La fiecare pas, citim un număr din fișier în variabila *x* și îl afișăm pe ecran.

```
while (!f.eof()) do
{
    f >> x;
    cout << x<<
}
```

Aplicație

R.VI.1. Șir de numere citite dintr-un fișier

Se citește un șir de numere întregi din fișierul "numere.txt". Fișierul conține: pe primul rând numărul *n* de elemente ale șirului, apoi, pe fiecare din următoarele *n* rânduri, câte un element. Să se afișeze suma elementelor pozitive ale șirului. *Exemplu:* dacă fișierul este cel din figură, atunci programul va tipări valoarea 15.

```
5
2
-3
8
0
-7
```

Rezolvare

Asignăm fișierului dat descriptorul *f* și îl deschidem pentru citire, cu atributul "r". Presupunând că fișierul a fost creat în directorul curent, instrucțiunea de asignare și deschidere este {*f=fopen("numere.txt", "r");*}. Citim valoarea lui *n* de pe primul rând din fișier {*fscanf(f, "%d\n", &n);*}. Inițializăm cu 0 suma *s* a elementelor pozitive din șir.

Cele *n* elemente ale șirului vor fi citite succesiv din fișier în aceeași variabilă *x*. Folosim un ciclu *for*, în care contorul *i* va număra elementele citite. Astfel, valorile lui *i* vor fi pe rând 1, 2, ..., *n*, indicând la al câta element al șirului am ajuns cu citirea (pentru *i*=1 citim primul element, apoi pentru *i*=2 al doilea element, etc). La fiecare pas:

- citim în variabila *x* un element al șirului, aflat pe un rând în fișier {*fscanf(f, "%d\n", &x);*};
- testăm dacă elementul tocmai citit în *x* este pozitiv, iar în caz afirmativ îl adăugăm la *s* {*if (x>0) s+=x;*}.

Programul se încheie cu închiderea fișierului și afișarea lui *s*.

```
#include <iostream.h>
#include <stdio.h>

FILE *f;
int i, x, n, s;
```

```
void main ()
```

```
{
    // asignare si deschidere fisier
    f=fopen("numere.txt", "r");
    fscanf(f, "%d\n", &n); // citește numărul de elemente de pe primul rand din fisier
    s=0;
    for (i=1; i<=n; i++) // citește pe rand cele n elemente, din fisier in variabila x
    {
        fscanf(f, "%d\n", &x);
        cout << x << " ";
        // pentru fiecare x, daca este pozitiv, atunci il adauga la suma S a elementelor pozitive
        if (x>0)
            s+=x;
    }
    cout << "\nSuma elementelor pozitive este " << s;
}
```

Încercați singuri!

► 1. (Bacalaureat iulie 2009, varianta 1)

Fișierul text **BAC.TXT** conține, pe o singură linie, cel mult 1000 de numere naturale nenule cu cel mult 4 cifre fiecare, numerele fiind separate prin câte un spațiu. Scrieți un program care citește de la tastatură un număr natural nenul *n* (*n*≤999) și numerele din fișierul **BAC.TXT** și care afișează pe ecran, separate prin câte un spațiu, toate numerele din fișier care sunt divizibile cu *n*. Dacă fișierul nu conține niciun astfel de număr, atunci se va afișa pe ecran mesajul **NU EXISTA**.

Exemplu: dacă fișierul **bac.txt** conține numerele:

3 100 40 70 25 5 80 6 3798

atunci pentru *n*=10 pe ecran se va afișa:

100 40 70 80

► 2. (Bacalaureat iulie 2009, varianta 42)

Fișierul text **NUMERE.TXT** conține pe prima linie un număr natural *n* (*0*<*n*<100000) iar pe doua linii, separate prin câte un spațiu, *n* numere naturale formate din cel mult două cifre.

a) Scrieți un program care determină în mod eficient, din punct de vedere al timpului de executare, dacă numerele situate pe a doua linie a fișierului sunt în ordine strict crescătoare. În caz afirmativ, programul va afișa pe ecran mesajul **DA**, altfel va afișa mesajul **NU**.

Exemplu: dacă fișierul **NUMERE.TXT** are următorul conținut:

7

3 5 2 1 5 23 1

atunci pe ecran se va afișa: **NU**

b) Descrieți succint, în limbaj natural, metoda de rezolvare folosită, explicând în ce constă eficiența ei (3 – 4 rânduri).

Aplicație R.VI.2. Intervale

(Bacalaureat iulie 2009, varianta 23)

Fișierul **BAC.TXT** conține pe prima linie un număr natural nenul n ($1 \leq n \leq 1000$), iar pe fiecare din următoarele n linii câte două numere întregi a și b ($1 \leq a \leq b \leq 32000$), fiecare pereche reprezentând un interval închis de forma $[a, b]$ pe axa numerelor. Scrieți un program care citește numerele din fișier, apoi determină un interval dintre cele citite care conține cel mai mare număr de valori întregi dintre toate intervalele formate. Programul va afișa pe un rând al ecranului, separate printr-un spațiu, capetele intervalului astfel determinat. În cazul în care există mai multe intervale care îndeplinesc această proprietate, se va tipări acela a cărei extremitate dreaptă este minimă în raport cu celelalte.

Exemplu: Dacă fișierul **BAC.TXT** are conținutul din figură, pe ecran se vor afișa valorile 6 și 13; numărul maxim de valori întregi conținute de intervalele din fișier este 8, există două intervale care cuprind în ele 8 valori întregi, respectiv $[17, 24]$ și $[6, 13]$, iar dintre acestea, intervalul $[8, 15]$ are capătul din dreapta mai mic.

4
17 24
-2 3
9 15
6 13

Rezolvare

Definim o variabilă nr , în care, la citirea fiecărui interval de forma $[a, b]$, memorăm numărul valorilor întregi din intervalul respectiv, care este $b - a + 1$ (de exemplu, intervalul $[17, 24]$ conține $24 - 17 + 1 = 8$ valori întregi). Mai avem nevoie de încă două variabile:

- $nrmax$ = numărul maxim de valori întregi dintre toate intervalele $[a, b]$, pe care îl inițializăm cu $-MAXINT$ (cel mai mic număr întreg posibil);
- $cmin$ = cel mai mic dintre capetele din dreapta ale intervalelor care conțin exact $nrmax$ valori întregi, pe care îl inițializăm cu $MAXINT$ (cel mai mare întreg posibil).

Asignăm descriptorul f fișierului **BAC.TXT** și îl deschidem pentru citire (cu parametrul $ios::in$ la funcția **fstream**). Apoi, de pe primul rând al fișierului, citim valoarea lui n , reprezentând numărul de intervale $\{f \gg n\}$.

Știind că avem de citit din fișier n rânduri, proiectăm un ciclu **for**, în care contorul i nu are alt rol decât acela de a număra citirile, parcurgând pe rând valorile $1, 2, \dots, n$. La fiecare pas trebuie să actualizăm, dacă este cazul, intervalul care îndeplinește ambele condiții din enunț, și care trebuie afișat. Pentru aceasta, vom parcurge în cadrul fiecărui pas următoarea succesiune de operații:

- ✓ Citim de pe un rând al fișierului, în variabilele a și b , cele două numere care reprezintă capetele unui interval de forma $[a, b]$ $\{f \gg a \gg b\}$;
- ✓ Calculăm numărul nr al valorilor întregi din intervalul $[a, b]$ astfel format $\{nr = b - a + 1\}$;
- ✓ Există două situații în care se modifică intervalul cu număr maxim de valori întregi și capătul din dreapta minim, și anume:
 - dacă numărul valorilor întregi din intervalul $[a, b]$ tocmai citit este mai mare decât numărul maxim ($nr > nrmax$)

SAU

- dacă am dat de un interval $[a, b]$ care conține un număr de valori întregi întâlnit și la alt interval, și capătul din dreapta al intervalului este mai mic decât capătul minim ($b < cmin$)

Într-un cuvânt, condiția care impune actualizarea intervalului $[a, b]$ cerut este: $(nr > nrmax) \ || \ (nr == nrmax \ \&\& \ b < cmin)$

Dacă este îndeplinită condiția de mai sus, înseamnă că intervalul $[a, b]$ curent îndeplinește ambele cerințe, deci:

- Noul capăt minim $cmin$ devine capatul b al intervalului curent $\{cmin = b\}$;
- Noul $nrmax$ devine numărul nr al valorilor întregi din intervalul $\{nrmax = nr\}$;
- Salvăm capetele a și b ale intervalului curent în variabilele $cap1$ și $cap2$, pentru că, cel puțin pentru moment, acesta este intervalul selectat ce trebuie afișat $\{cap1 = a; \ cap2 = b\}$.

```
for (i=1; i<=n; i++)
{
    f >> a >> b;
    cout << endl << "[" << a << ", " << b << "]";
    nr=b-a+1;
    if ((nr>nrmax) || (nr==nrmax && b<cmin))
    {
        cmin=b;
        nrmax=nr;
        cap1=a;
        cap2=b;
    }
}
```

La finele ciclului, în variabila $nrmax$ avem numărul maxim de valori întregi dintre toate intervalele, iar în $cap1$ și $cap2$ am salvat capetele intervalului cu $nrmax$ valori întregi, care în plus mai au și capătul din dreapta minim. Ca atare, programul se încheie cu afișarea acestor valori și închiderea fișierului f .

```
// Bac 2009, II.4 / var 29
#include <iostream.h>
#include <conio.h>
#include <fstream.h>
#include <values.h>
```

```
void main ()
{
    clrscr();
    int n,i,a,b,nr,nrmax,cmin,cap1,cap2;
    fstream f("bac.txt",ios::in);
    f >> n; // citește numărul n de intervale
    nrmax=-MAXINT; // nrmax = numărul maxim de valori întregi aflate în intervalele date
    cmin=MAXINT;
    // cmin = capatul din dreapta cel mai mic dintre interv. care contin nrmax valori întregi
    cout << "Intervalele citite din fisier sunt: ";
    // citește cele n intervale, și, pentru fiecare interval, actualizează nrmax și cmin
```



```

for (i=1; i<=n; i++)
{
    f >> a >> b;
    cout << endl << "[" << a << ", " << b << "]";
    nr=b-a+1;
    if ((nr>nrmax) || (nr==nrmax && b<cmin))
    {
        cmin=b;
        nrmax=nr;
        // in cap1 si cap2 actualizam capetele intervalului care indeplineste cele doua conditii
        cap1=a;
        cap2=b;
    }
}
cout << endl << "Intervalul este [" << cap1 << ", " << cap2 <<
"]";
cout << endl << "Si contine " << nrmax << " valori intregi";
f.close();
getch();
}

```

Încercați singuri!

► 1. Fișierul text "triunghi.txt" conține pe fiecare linie câte trei numere separate prin câte un spațiu. Nu se cunoaște câte astfel de triplete sunt în fișier. Pentru fiecare dintre aceste triplete să se verifice dacă pot reprezenta laturile unui triunghi (sunt pozitive și suma a oricare două este mai mare decât al treilea număr); în caz afirmativ se va stabili natura triunghiului: isoscel, echilateral, oarecare.

► 2. (Bacalaureat iulie 2009, varianta 81)

Fișierul text NUMERE.IN conține, pe fiecare linie a sa, câte două numere naturale mai mici sau egale decât 10000, despărțite printr-un spațiu. Scrieți un program care să afișeze pe ecran, unul sub altul, cel mai mare număr de pe fiecare linie, ca în exemplu.

Exemplu: dacă fișierul NUMERE.IN are conținutul din figură:

atunci pe ecran se afișează:

14
110
4

12 14
110 12
4 -8

Aplicație

R.VI.3. Sortarea unui șir de numere citit din fișier

(Bacalaureat august 2000, varianta 2)

În fișierul text "BAC.TXT" se află mai multe numere naturale de cel mult trei cifre fiecare, numere despărțite între ele prin câte un spațiu. Scrieți un program care crează un alt fișier text numit "BAC2.TXT" care să conțină aceleași numere din fișierul "BAC.TXT", câte unul pe linie, în ordinea crescătoare a valorii acestora.

Rezolvare

Asignăm fișierelor "BAC.TXT" și "BAC2.TXT" descriptorii *f* respectiv *g*. Deschidem fișierul *f* pentru citire (cu atributul "r") și fișierul *g* pentru scriere (cu atributul "w").

Citim șirul de numere din fișier într-un vector *v*, folosind un ciclu *while*. La fiecare pas citim un număr din fișier, adăugându-l ca element nou la sfârșitul vectorului.

Notăm cu *i* poziția la care am ajuns cu adăugarea în vector. Evident valoarea inițială a lui *i* este 0, pentru că la început vectorul este gol și primul element care se va adăuga în el va ocupa poziția 0. Cât timp nu am ajuns la sfârșitul rândului în fișier {not seekeoln(*f*)}:

- citim un număr din fișier, în elementul *v[i]* al vectorului {*fscanf(f, "%d", &v[i]);*}
- "pregătim" poziția următoare la sfârșitul vectorului *v*, poziție pe care se va memora numărul ce va fi citit la pasul următor al ciclului {*i++;*}

Valoarea finală a lui *i* reprezintă numărul de elemente ale vectorului {*n=i;*}.

```

i=0;
while (!feof(f))
{
    fscanf(f, "%d", &v[i]);
    cout << v[i] << endl;
    i++;
}
n=i;

```

Exemplu:

Presupunem că fișierul conține șirul 12, -3, 8. Inițial *i*=0.

Pasul 1: end of file ? *nu* ⇒ citește *v[0]*=12, *i*=*i*+1=0+1=1;

Pasul 2: end of file ? *nu* ⇒ citește *v[1]*=-3, *i*=*i*+1=1+1=2;

Pasul 3: end of file ? *nu* ⇒ citește *v[2]*=8, *i*=*i*+1=2+1=3;

Pasul 4: end of file? *da* ⇒ iese din ciclu.

n=i ⇒ *n*=3 numărul de elemente ale vectorului.

Sortăm crescător vectorul *v*. Reamintim pe scurt algoritmul de sortare:

- Fiecare din elementele *v[0], v[1], ..., v[n-2]* va juca pe rând rolul unui așa numit pivot. Parcurgem pivoții într-un ciclu în care contorul *i* ia pe rând valorile 0, 1, ..., *n-2* și la fiecare pas al ciclului, comparăm pivotul *v[i]* cu toate elementele aflate după el, *v[i+1], ..., v[n-1]*. În acest scop, parcurgem pozițiile acestor elemente într-un alt ciclu cu *j=i+1, ..., n-1*, și pentru fiecare element *v[j]*, dacă este mai mare decât pivotul *v[i]*, atunci inter schimbăm *v[j]* cu *v[i]* prin "metoda paharelor".

```

for (i=0; i<=n-2; i++)
    for (j=i+1; j<=n-1; j++)
        if (v[j]<v[i])
        {
            temp=v[i];    v[i]=v[j];    v[j]=temp;
        }

```

❏ Scriem vectorul sortat în fișierul *g*, fiecare element pe câte un rând. Parcurgem într-un ciclu pozițiile $i=0, 1, \dots, n-1$ ale elementelor vectorului, și la fiecare pas scriem elementul $v[i]$ în fișierul *g* (cu parametrul de control "%d\n", pentru a se asigura scrierea fiecărui element pe câte un rând).

```
for (i=0; i<=n-1; i++)
    fprintf(g, "%d\n", v[i]);
```

❏ În final închidem cele două fișiere, cu procedurile *fclose(f)* respectiv *fclose(g)*.

```
#include <iostream.h>
#include <stdio.h>

FILE *f,*g;
int n,i,j,temp,v[20];

void main ()
{
    // citește elementele vectorului din fișier, într-un ciclu
    f=fopen("C:\\BAC.TXT", "r");
    g=fopen("C:\\BAC2.TXT", "w");
    i=0;
    while (!feof(f))
    {
        fscanf(f, "%d\n", &v[i]);
        cout << v[i] << endl;
        i++;
    }
    n=i;
    // sortează crescător vectorul
    for (i=0; i<=n-2; i++)
        for (j=i+1; j<=n-1; j++)
            if (v[j]<v[i])
            {
                temp=v[i];
                v[i]=v[j];
                v[j]=temp;
            }
    // scrie vectorul sortat în fișierul de ieșire
    for (i=0; i<=n-1; i++)
        fprintf(g, "%4d", v[i]);
    fclose(f); fclose(g);
    cout << "\nS-a scris vectorul sortat în fișierul vect.out";
}
```

❏ Încercați singuri!

► 1. (Bacalaureat iulie 2009, varianta 99 – enunț adaptat)

Pe prima linie a fișierului text *DATE.TXT* se află un număr natural nenul n ($n \leq 1000$), iar pe a doua linie un șir de n numere întregi nenule, depărțite prin câte un spațiu, fiecare având cel mult nouă cifre.

a) Scrieți un program care citește numerele din fișier și ordonează descrescător doar numerele pozitive din șir, fără a modifica pozițiile numerelor negative. Programul va afișa, pe ecran, pe o singură linie, șirul obținut după ordonare, numerele fiind despărțite prin câte un spațiu. În locul fiecărui număr negativ din șirul citit se va afișa valoarea 0. Realizați un program eficient din punct de vedere al spațiului de memorie folosit.

Exemplu: dacă fișierul *DATE.TXT* conține:

7

32 -491 23 -328 213 51 -4

pe ecran se va afișa: 213 0 51 0 32 23 0

b) Descrieți succint, în limbaj natural, metoda utilizată, justificând eficiența acesteia.

❏ Aplicație R.VI.4. Numere din fișier cu prima cifră pară

(Bacalaureat iulie 2008, varianta 88)

Fișierul *DATE.IN* conține cel mult 100000 de numere naturale, scrise toate pe un rând și separate prin spații, fiecare număr având cel mult nouă cifre. Să se realizeze un program care scrie în fișierul *DATE.OUT*, pe o singură linie, separate prin câte un spațiu, toate numerele din fișierul *DATE.IN* care au prima cifră pară. (prima cifră a unui număr este considerată cea mai semnificativă, adică cea mai din stânga).

Exemplu:

dacă fișierul *DATE.IN* conține numerele

45 123 68 8 134 56 876 6666 2 5 123 65

atunci în fișierul *DATE.OUT* se vor găsi următoarele valori:

45 68 8 876 6666 2 65

❏ Rezolvare

❏ Mai întâi asignăm descriptorul *f* fișierului '*DATE.IN*' și îl deschidem pentru citire cu parametrul "*ios.in*" (*fstream f("date.in", ios::in);*), precum și descriptorul *g* fișierului '*DATE.OUT*', pe care îl deschidem pentru scriere cu parametrul "*ios.out*" (*fstream g("date.out", ios::out);*).

❏ Vom citi succesiv numerele din fișierul *f*, în aceeași variabilă *x*. Întrucât nu știm câte valori se găsesc în fișier, încheierea citirii se va produce în momentul în care am ajuns la sfârșitul rândului pe care se află toate numerele, care de fapt coincide cu sfârșitul fișierului. Așadar, cât timp cât nu s-a atins sfârșitul de fișier (adică atâta timp cât funcția *f.eof()* returnează 0 (condiție care se poate scrie sub forma "*!f.eof()*"), executăm următoarele acțiuni:

- ✓ Citim un număr din fișier în variabila *x* (cu *cin*);
- ✓ Extragem cifrele lui *x* cu algoritmul cunoscut, pentru a ajunge la prima cifră; inițializăm un deîmpărțit *d* cu numărul *x*, apoi proiectăm un ciclu de împărțiri succesive ale lui *d* la 10, care se execută atâta timp cât *d* este diferit de 0; la fiecare pas, extragem o cifră în variabila *c* (ca fiind restul împărțirii

lui d la 10), apoi pregătim împărțirea următoare (noul deîmpărțit d va fi câtul împărțirii curente, $d \div 10$);

```
d=x;
while (d)
{
    c=d%10;
    d/=10;
}
```

- ✓ După ce am încheiat extragerea cifrelor numărului citit în variabila x la un pas al primului ciclu, avem certitudinea că ultima cifră extrasă, adică valoarea lui c cu care s-a ieșit din ciclu, reprezintă prima cifră a lui x ; testăm dacă această cifră este pară (adică dacă restul împărțirii lui c la 2 este 0), iar în caz afirmativ scriem în fișierul g respectiva cifră c , urmată de un spațiu (pentru că toate valorile care îndeplinesc condiția cerută trebuie scrise pe un singur rând al fișierului de ieșire).

```
if (!(c%2))
    g << x << " ";
```

În final, mai afișăm pe ecran un mesaj care anunță construirea fișierului DATE.OUT, și închidem fișierele f și g .

```
// Bac 2008, varianta 88
#include <iostream.h>
#include <conio.h>
#include <fstream.h>

void main ()
{
    clrscr();
    long x,d;
    int c;
    fstream f("f:\\Fisiere\\date.in",ios::in);
    fstream g("f:\\Fisiere\\date.out",ios::out);
    while (!f.eof()) // cat timp nu am ajuns la sfarsitul fisierului
    {
        // citeste un numar din fisierul f in variabila x
        f >> x;
        // extrage cifrele lui x
        d=x;
        while (d)
        {
            c=d%10;
            d/=10;
        }
        // daca ultima cifra extrasa (prima din numar) este para, scrie numarul x in fisierul g
        if (!(c%2))
            g << x << " ";
    }
    cout << endl << "S-au scris numerele cerute in fisierul date.out";
    f.close();
    g.close();
}
```

Încercați singuri!

► 1. (Bacalaureat iunie 2008, varianta 72)

Pe prima linie a fișierului text BAC.IN se află un număr natural n ($0 < n \leq 1000$), iar pe a doua linie n numere reale pozitive, despărțite prin câte un spațiu. Realizați un program care citește datele din fișierul de intrare, apoi scrie în fișierul BAC.OUT, pe o singură linie, despărțite prin câte un spațiu, acele numere din fișierul de intrare a căror parte întreagă este număr prim. Dacă nu există nici o valoare în fișierul BAC.IN cu proprietatea cerută, atunci se va afișa mesajul NU.

Exemplu: dacă fișierul BAC.IN are conținutul din figură,

6
12.095 31.567 5.789 789.834 1234.923 2.345

atunci după execuția programului, în fișierul BAC.OUT se găsește următorul șir de numere: 31.567, 5.789, 2.345

► 2. (Bacalaureat iunie 2008, varianta 74)

Fișierul BAC.IN conține pe prima linie un număr natural n ($0 < n \leq 5000$), iar pe a doua linie n numere naturale de cel mult nouă cifre fiecare, separate prin câte un spațiu. Să se scrie un program care citește n , apoi cele n numere naturale din fișierul BAC.IN și scrie în fișierul text BAC.OUT, pe câte o linie fiecare, numerele de pe a doua linie a fișierului BAC.IN care au exact patru cifre.

Exemplu:

BAC.IN	BAC.OUT
6	1124
23 1124 567 89013 5586 1200	5586
	1200

Aplicație R.VI.5. Fișiere identice

Scrieți un program care verifică dacă două fișiere date sunt identice sau nu (conțin exact aceleași caractere) și afișează un mesaj corespunzător.

Rezolvare

Folosim o variabilă "semafor" e , a cărei valoare va indica "starea" condiției testate. Astfel, valoarea finală a lui e va fi: 1 dacă fișierele cu descriptorii f și g sunt identice, respectiv 0 în caz contrar. Aplicăm metoda reducerii la absurd: presupunem inițial că fișierele sunt identice inițializând e cu 1, apoi încercăm să găsim cazul contrar.

Într-un ciclu, citim "în paralel" fișierele, caracter cu caracter. Cât timp e este 1 și nu am ajuns la sfârșitul unuia din fișiere $\{!(feof(f)) || !(feof(g))\}$

- ✓ citește un caracter $c1$ din fișierul f și un caracter $c2$ din fișierul g ;
- ✓ dacă găsește o pereche de caractere ($c1, c2$) diferite, atunci fișierele nu mai pot fi identice, deci e devine 0.

☞ Dacă până acum fișierele sunt identice, mai trebuie testată o posibilitate. Așa cum am proiectat ciclul, nu rezultă de nicăieri dacă ieșirea din el s-a produs din cauză ca s-a atins sfârșitul ambelor fișiere, sau urmare a faptului că s-a ajuns doar la sfârșitul unuia dintre ele. A doua situație apare atunci când unul din fișiere conține niște caractere în plus față de celălalt, caz în care fișierele nu mai sunt identice, chiar dacă caracterele testate coincid. Acest caz se analizează separat. Dacă e sfârșitul lui *f* și nu e sfârșitul lui *g* {*feof(f) && !feof(g)*}, atunci variabila *e* ia valoarea 0. Același lucru se întâmplă și în situația inversă: este sfârșitul lui *g* dar nu este sfârșitul lui *f*.

☞ În final, testăm valoarea variabilei *e* și în funcție de aceasta tipărim un mesaj.

```
# include <stdio.h>
# include <iostream.h>

char c1,c2;
FILE *f,*g;
int e;

void main ()
{
    f=fopen("c:\\1.txt","r");
    g=fopen("c:\\2.txt","r");
    e=1; // in final, e va fi 1 daca fisierul sunt identice, respectiv 0 in caz contrar
    // citeste cate un caracter din cele doua fisier si verifica egalitatea caracterelor
    while (e && !(feof(f) || feof(g)))
    {
        c1=fgetc(f);
        c2=fgetc(g);
        if (c1!=c2)
            e=0;
    }
    // daca pana acum fisierul coincid, e posibil ca unul din ele sa aiba caractere in plus fata de celalalt
    if (e)
        if (feof(f) && !feof(g))
            e=0;
        if (!feof(f) && feof(g))
            e=0;
    fclose(f);
    fclose(g);
    if (e)
        cout << "\n Fisiere identice";
    else
        cout << "\n Fisiere diferite";
}
```

Încercați singuri!

► 1. (Bacalaureat iunie 2004, varianta 1)

Scrieți programul care creează fișierul text **BAC.TXT** ce conține pe prima sa linie, în ordine, toate literele mari ale alfabetului englez aflate în alfabet după o literă mare dată de la tastatură.

Exemplu: dacă se citește litera **R**, atunci **BAC.TXT** va conține: **STUVWXYZ**

► 2. (Bacalaureat iunie 2004, varianta 1)

Fișierul text **BAC.TXT** conține, pe prima linie, un șir de exact 1000 de caractere (litere, cifre și spații). Scrieți un program care citește de la tastatură un număr natural *n* ($0 < n \leq 1000$) și afișează pe ecran câte caractere cifră există printre primele *n* caractere din fișier.

De exemplu, dacă **BAC.TXT** conține: **mx 12 5 3a63abc12 fcccc...c** (unde litera **c** apare la sfârșit de 981 de ori) iar de la tastatură se citește valoarea *n*=11, atunci se va afișa pe ecran numărul 5, deoarece în șirul format din primele 11 caractere există cinci caractere cifră (cele subliniate).



► 1. (Bacalaureat iulie 2009, varianta 3)

Fișierului text **NR.TXT** conține pe o singură linie, separate prin câte un singur spațiu, cel mult 100 de numere naturale, fiecare număr având cel mult 4 cifre. Să se scrie un program care citește toate numerele din fișier și scrie pe ecran, pe o singură linie, separate prin câte un spațiu, în ordine crescătoare, toate numerele din fișier care au cel puțin trei cifre. Dacă fișierul nu conține astfel de numere se va afișa mesajul **NU EXISTA**.

Exemplu: dacă fișierul conține numerele: 123 17 3844 459 9 68 atunci pe ecran se vor afișa valorile: 123 3844 459

► 2. (Bacalaureat iulie 2009, varianta 2)

Fișierul text **NR.TXT** conține pe o singură linie, separate prin câte un singur spațiu, cel mult 100 de numere întregi, fiecare număr având cel mult 4 cifre. Scrieți un program care citește numerele din fișierul **NR.TXT** și afișează pe ecran, separate prin câte un spațiu, în ordine crescătoare, toate numerele *nenule* din fișier. Dacă nu există astfel de numere se va afișa mesajul **NU EXISTA**.

Exemplu: dacă fișierul **NR.TXT** conține numerele:

-3 -10 0 7 -5 7 51 -800 6 3798,

atunci pe ecran se va afișa: 7 7 51 6 3798

Indicații: Într-un ciclu, cât timp nu am ajuns la sfârșitul fișierului, citim succesiv numerele din fișier în aceeași variabilă *x*. La fiecare pas, pentru fiecare *x* citit, ca să verificăm dacă valoarea lui *x* este un număr natural, este suficient să testăm dacă $x > 0$ și $x = [x]$.

► 3. (Bacalaureat iulie 2009, varianta 2..)

Fișierul text **BAC.TXT** conține pe prima linie numărul natural *n*, $1 \leq n \leq 10000$, pe următoarele *n* linii un tablou unidimensional de *n* numere întregi, iar pe ultima linie două numere întregi *a* și *b* ($a \leq b$) separate de un spațiu. Fiecare dintre cele *n* numere, precum și valorile *a* și *b*, au cel mult patru cifre.

Scrieți un program care afișează pe ecran cel mai mic număr natural din intervalul închis $[a, b]$ care se găsește în șirul dat. Dacă nu există un astfel de număr, programul afișează textul NU.

Exemplu: dacă fișierul BAC.TXT are conținutul alăturat, programul afișează valoarea 14.

```
4
-2
72
14
25
8 35
```

► 4. (Bacalaureat iulie 2000, varianta 2)

Scrieți un program care generează toate numerele prime strict mai mici decât x (x număr natural). Valoarea variabilei x se citește de la tastatură. Numerele prime generate vor fi scrise în fișierul text BAC.TXT, câte unul pe linie.

Exemplu: pentru $x=10$ se vor scrie în fișier numerele 2, 3, 5 și 7.

► 5. Se citesc de la tastatură două numere întregi a și b . Realizați un program care crează un fișier text ce va conține, pe un singur rând separate prin spații, toate numerele naturale cuprinse între a și b alcătuite numai din cifre identice.

Exemplu:

Pentru $a=10$ și $b=250$, fișierul de ieșire va arăta astfel:

```
11 22 33 44 55 66 77 88 99 111 222
```

Indicații: Într-un ciclu, parcurgem în contorul i toate numerele naturale de la a la b , și pentru fiecare i testăm dacă este format numai din cifre identice, iar în caz afirmativ scriem în fișier respectiva valoare a lui i . Pentru a verifica proprietatea cerută, folosim metoda reducerii la absurd, adică presupunem inițial că i este alcătuit numai din cifre identice, inițializând cu 1 o variabilă ok cu sens logic, apoi căutăm cazul contrar. În acest scop extragem cifrele lui i și dacă am găsit două cifre alăturate diferite între ele, atunci ok ia valoarea 0.

► 6. (Bacalaureat iulie 2009, varianta 5)

Scrieți un program care citește de la tastatură un număr natural n cu cel mult 8 cifre ($n \leq 10$) și care creează fișierul text NR.TXT ce conține numărul n și toate prefixele nenule ale acestuia, pe o singură linie, separate prin câte un spațiu, în ordine descrescătoare a valorii lor.

Exemplu: pentru $n=10305$ fișierul NR.TXT va conține numerele:
10305 1030 103 10 1

► 7. (Bacalaureat iulie 2001, varianta 4)

Scrieți un program Pascal care citește de la tastatură un număr n ($n \leq 10^9$) și creează fișierul text BAC.TXT, care să conțină toate pătratele perfecte mai mici sau egale cu n , despărțite între ele prin câte un spațiu. Algoritmul corespunzător programului trebuie să aibă în vedere o prelucrare cât mai eficientă ca timp de executare și spațiu de memorie utilizat.

Exemplu: pentru $n=68$, conținutul fișierului BAC.TXT este următorul:

```
0 1 4 9 16 25 36 49 64
```

► 8. (Bacalaureat iulie 2009, varianta 8)

Scrieți un program care citește de la tastatură un număr natural n ($n \leq 50$) și care afișează pe prima linie a fișierului BAC.TXT primele n numere pare, pe a doua linie primele $n-1$ numere pare, ..., pe linia n primul număr par. Numerele vor fi afișate pe fiecare linie în ordine crescătoare și vor fi separate prin câte un spațiu.

```
0 2 4
0 2
0
```

Exemplu: dacă $n=3$ fișierul BAC.TXT va avea conținutul din figură.

Indicații: Folosim două cicluri. În primul, contorul i va număra rândurile de la 1 la n , iar în al doilea contorul j va trece în revistă numerele afișate pe fiecare linie i , respectiv primele i numere naturale pare. Dacă în cazul primului ciclu putem folosi un `for`, pentru al doilea ne va trebui un `while` (inițial $j=0$, ciclul se execută cât timp $j \leq i$, și la fiecare pas facem $j=j+2$).

► 9. (Bacalaureat iulie 2001, varianta 6)

Scrieți un program Pascal care citește de la tastatură un număr natural n , ($0 < n \leq 15$) și crează un fișier text cu numele 'A.TXT' ce conține pe prima linie un caracter A , pe a doua linie două caractere A nedespărțite prin spații, pe linia a treia trei caractere A nedespărțite prin spații etc. Ultima linie a fișierului trebuie să fie linia a n -a care să conțină n caractere A nedespărțite prin spații.

Exemplu:

dacă se citește $n=4$, fișierul 'A.TXT' creat trebuie să fie:

```
A
AA
AAA
AAAA
```

► 10. (Simulare Bacalaureat, iulie 2005)

Scrieți programul Pascal care citește de la tastatură un număr natural nenul n și creează fișierul text BAC.TXT ce conține: pe prima linie n numere de 1, pe a doua linie $n-1$ numere egale cu 2, pe a treia linie $n-2$ numere egale cu 3, etc., pe ultima linie conținând o singură dată numărul n . De exemplu, dacă se citește $n=4$, atunci 'bac.txt' va avea conținutul din figură.

```
1 1 1 1
2 2 2
3 3
1
```

► 11. (Bacalaureat iulie 2008, varianta 84)

Scrieți un program care citește de la tastatură un număr natural nenul n , apoi construiește fișierul STELUTE.TXT, astfel încât acesta să conțină: pe prima linie un singur caracter '*', pe a doua linie două caractere '*', ..., pe a n -a linie n caractere '*', apoi, în continuare $n-1$ caractere '*' pe linia $n+1$, $n-2$ caractere '*' pe linia $n+2$, ..., ultima linie, cea cu numărul de ordine $2n-1$ conținând, la fel ca și prima, un singur caracter '*'.

Exemplu: pentru $n=4$, fișierul va avea conținutul din figură:

```
*
**
***
****
***
**
*
```

► 12. (Bacalaureat iulie 2009, varianta 58)

Fișierul text BAC.TXT conține, pe o singură linie, cel puțin două și cel mult 100 de numere naturale nenule distincte de cel mult patru cifre fiecare, numerele fiind separate prin câte un spațiu. Scrieți un program care citește numerele din fișier și scrie pe ecran, în ordine crescătoare, cele mai mici două numere dintre cele citite. Se cere un algoritm eficient de rezolvare.

Exemplu: dacă fișierul **BAC.TXT** conține numerele:

1017 48 310 5710 162

atunci se va afișa: 48 162

Indicații: Un algoritm eficient trebuie să evite memorarea numerelor într-un vector, motiv pentru care le vom citi pe rând într-un ciclu, în aceeași variabilă x , cât timp nu am ajuns la sfârșitul fișierului. Notăm cu min1 cel mai mic număr din fișier iar cu min2 al doilea număr ca mărime, și inițializăm ambele variabile cu MAXINT . Pentru fiecare x citit, facem două testări:

– dacă x este mai mic decât min1 , atunci el devine noul min1 ;

– dacă $x > \text{min1}$ dar în același timp $x < \text{min2}$, atunci x devine noul min2 .

► **13. (Bacalaureat iulie 2009, varianta 15)**

În fișierul text **BAC.IN** se găsesc, pe o singură linie, separate prin câte un spațiu, mai multe numere naturale de cel mult șase cifre fiecare. Se cere să se determine și să se afișeze pe ecran ultimul număr impar din fișierul **BAC.IN**. Dacă în fișier nu există niciun număr impar se va scrie pe ecran mesajul **Nu există numere impare**.

Exemplu: dacă fișierul **BAC.IN** conține valorile: 12 6 25 68 13 8 24 31 42 se va afișa 31.

a) Descrieți în limbaj natural un algoritm eficient, din punct de vedere al spațiului de memorie și al timpului de executare, pentru rezolvarea acestei probleme, explicând în ce constă eficiența acestuia.

b) Scrieți programul corespunzător algoritmului descris.

► **14. (Bacalaureat iulie 2009, varianta 14)**

Fișierul text **BAC.TXT** conține mai multe numere naturale cu cel mult șase cifre fiecare, câte un număr pe fiecare linie a fișierului. Scrieți un program care afișează pe ecran toate numerele din fișier, câte cinci numere pe fiecare linie, separate prin câte un spațiu, cu excepția ultimei linii care poate conține mai puțin de 5 numere.

Exemplu: dacă fișierul conține numerele:

2 8 4 7 3 9 5 4 1 8 2 4

atunci pe ecran se vor afișa următoarele trei rânduri:

2	8	4	7	3
9	5	4	1	8
2	4			

Indicații: Cerându-se doar afișarea succesivă a numerelor, le putem citi pe rând în aceeași variabilă x , până la întâlnirea sfârșitului de fișier, fără a avea nevoie de un vector. Ca să afișăm câte cinci numere pe un rând avem nevoie de un contor care să numere de la 1 la 5, și pe care, de fiecare dată când ajunge la valoarea 5, îl reinițializăm cu 1.

► **15. (Bacalaureat iulie 2009, varianta 80)**

Fișierul text **BAC.IN** conține cel mult 1000 de numere naturale cu cel mult patru cifre fiecare, despărțite prin câte un spațiu. Scrieți un program care citește numerele din fișier și afișează pe ecran, în ordine crescătoare, acele numere din fișier care au cifrele egale. Dacă fișierul nu conține niciun astfel de număr, se va afișa pe ecran mesajul **NU EXISTA**.

Exemplu: dacă fișierul **BAC.IN** conține numerele: 30 44 111 7 25 5
atunci pe ecran se va afișa 5 7 44 111.

► **16. (Bacalaureat iulie 2009, varianta 47)**

Fișierul text **BAC.TXT** conține pe prima linie un număr natural n ($n < 100$), iar pe a doua linie, separate prin câte un spațiu, n numere naturale, mai mici decât 30000 fiecare. Scrieți un program care citește de la tastatură un număr natural k ($k < 10$) precum și numerele din fișierul **BAC.TXT** apoi determină și afișează pe ecran, cu câte un spațiu între ele, toate numerele de pe a doua linie a fișierului care conțin în scrierea lor cifra memorată în variabila k . Dacă nu există un asemenea număr pe a doua linie a fișierului, se afișează pe ecran mesajul **NU**.

Exemplu: dacă se citește de la tastatură $k=2$, iar fișierul are conținutul din imagine, atunci pe ecran se afișează numerele: 234 202 427 92

8
234 5678 317 809 202 427 92 6004

► **17. (Bacalaureat iulie 2009, varianta 94)**

Fișierul text **BAC.TXT** conține pe fiecare linie câte un număr întreg format din cel mult patru cifre. Se știe că fișierul conține cel puțin un număr. Scrieți un program eficient atât din punct de vedere al timpului de executare cât și din punct de vedere al spațiului de memorie utilizat, care citește de la tastatură un număr real x , apoi determină și afișează acel număr din fișierul **BAC.TXT** care are valoarea cea mai apropiată de valoarea lui x .

Exemplu: dacă fișierul **BAC.TXT** are conținutul alăturat, iar de la tastatură se citește valoarea -3.85 , programul va afișa valoarea -5 .

a) Descrieți succint, în limbaj natural, metoda de rezolvare folosită, explicând în ce constă eficiența ei (3 – 4 rânduri).

b) Scrieți un program C/C++ care rezolvă problema conform metodei descrise.

4
-5
-6
-984
1345
-1

► **18. (Bacalaureat iulie 2009, varianta 97)**

Pe prima linie a fișierului text **DATE.TXT** se află un număr natural n ($0 < n \leq 10000$), iar pe a doua linie un șir de n numere naturale, depărțite prin câte un spațiu, fiecare având cel mult patru cifre.

a) Scrieți un program care citește numerele din fișier și afișează, pe ecran, valorile din șir, în ordinea crescătoare a cifrei unităților. Dacă două numere din șir au aceeași cifră a unităților nu contează care dintre ele va fi afișat primul. Realizați un program eficient din punct de vedere al timpului de executare.

Exemplu: dacă fișierul **DATE.TXT** conține:

7
32 491 26 328 213 500 422

atunci pe ecran se va afișa: 500 491 32 422 213 26 328

b) Descrieți succint, în limbaj natural, metoda utilizată, justificând eficiența acesteia.

► 19. (Bacalaureat iulie 2009, varianta 89)

Fișierul text **BAC.TXT** conține pe mai multe rânduri cel mult 50000 de numere naturale din intervalul închis $[0, 99]$, numerele de pe același rând fiind separate prin câte un spațiu.

a) Scrieți un program care afișează pe ecran, în ordine crescătoare, acele numere din fișier care au cel puțin k divizori, k fiind un număr natural ($k < 20$) citit de la tastatură, utilizând un algoritm eficient din punct de vedere al timpului de executare. Dacă un număr care corespunde cerinței apare de mai multe ori, se va afișa o singură dată. Numerele vor fi afișate pe ecran separate prin câte un spațiu.

Exemplu: dacă fișierul conține numerele: 15 36 33 36 1 19 1 24 2, iar pentru k se citește valoarea 5, se vor afișa numerele 12 24 36.

b) Descrieți succint, în limbaj natural, algoritmul utilizat, justificând eficiența acestuia.

► 20. Se citește o matrice a din fișierul **MATRICE.TXT**. Fișierul conține:

- pe primul rând două numere întregi m și n , separate prin câte un spațiu;
- pe fiecare din următoarele m rânduri, cel n elemente ale unei linii a unei matrice a cu m linii * n coloane;

Realizați un program care scrie în fișierul **DIAG.TXT**, pe un singur rând separate prin spații, elementele pare de sub diagonala principală a matricii a , ordonate crescător.

Indicații: Citim mai întâi valorile lui m și n de pe primul rând al fișierului, apoi, într-un ciclu cu $i=1, 2, \dots, m$, parcurgem rândurile din fișier, care sunt de fapt liniile matricii. Pentru fiecare rând i , parcurgem coloanele j de la 1 la i , deoarece ne interesează doar elementele de sub diagonala principală. Pe acestea trebuie să le adăugăm pe rând într-un vector v , pentru a le putea sorta crescător. Un al treilea contor k , inițializat cu 1, va urmări poziția la care am ajuns cu adăugarea la sfârșitul vectorului v . Inițial $k=1$, și pentru fiecare element de sub diagonala principală, după ce l-am copiat în vector, facem $k++$ (pregătim următoarea "căsuță" în vector). După încheierea celor două cicluri de parcurgere, mai trebuie să sortăm vectorul v , folosind unul din algoritmii învățați. Dacă sunteți ingenioși, vă veți da seama că nici măcar nu este necesar să completăm în memorie matricea!

► 21. Fișierul **MAT.IN** conține:

- pe primul rând două numere întregi m și n , separate prin câte un spațiu;
- pe fiecare din următoarele m rânduri, cel n elemente ale unei linii a unei matrice a cu m linii * n coloane;

Realizați un program care construiește și scrie în fișierul **MAT.OUT** transpusa matricii a , adică matricea obținută din a prin inversarea liniilor și coloanelor între ele (liniile devin coloane, iar coloanele devin linii). Matricea transpusă se va scrie în fișierul **MAT.OUT** în același format ca și cea din fișierul de intrare.

Exemplu:

MAT.IN	MAT.OUT
2 3 -1	2 7
7 -5 9	3 -5
	-1 9

► 22. Fișierul **TABLOU.IN** conține m rânduri de numere întregi, pe fiecare rând găsim-se m numere separate prin spații. Interpretând conținutul fișierului ca un "tablou" cu m linii * n coloane de numere, realizați un program care mută la sfârșitul partea dreaptă a tabloului acele coloane care conțin cel puțin p elemente nule (unde valoarea lui p se citește de la tastatură). Coloanele care îndeplinesc condiția vor fi mutate exact în ordinea în care sunt parcurse de la stânga la dreapta, iar celelalte coloane vor ocupa partea de început a tabloului, exact în ordine inițială. Tabloul rezultat după mutare se va scrie în același format în fișierul **TABLOU.OUT**.

Exemplu: pentru $p=3$ și fișierul **TABLOU.IN** de mai jos, noua ordine a coloanelor în fișierul **TABLOU.OUT** va fi (2 4 5 1 3); coloanele 1 și 3 au fost mutate la sfârșit deoarece conțin cel puțin trei elemente nule.

TABLOU.IN	TABLOU.OUT
2 1 0 0 1 9	1 0 1 9 2 0
0 0 0 3 1 2	0 3 1 2 0 0
0 7 4 3 1 7	7 3 1 7 0 4
0 0 0 3 1 0	0 3 1 0 0 0
5 2 0 6 1 0	2 6 1 0 5 0

► 23. (Bacalaureat iulie 2009, varianta 51)

Fișierul text **DATE.IN** conține pe prima linie un număr natural nenul n ($n \leq 100$), iar pe a doua linie n numere naturale nenule separate prin câte un spațiu. Scrieți un program care citește toate numerele din fișier, apoi construiește în memorie un tablou unidimensional a , ale cărui elemente vor fi cele n valori citite, și afișează pe ecran perechile de elemente de forma (a_i, a_j) , unde $1 \leq i < j \leq n$, cu proprietatea că ambele elemente ale perechii au aceeași paritate. Fiecare astfel de pereche se va afișa pe câte o linie a ecranului, cu elementele sale separate prin câte un spațiu. În cazul în care nu există nici o pereche cu proprietatea cerută, se va afișa valoarea 0.

Exemplu: dacă fișierul **DATE.IN** are conținutul din figură, atunci pe ecran se vor afișa perechile: (16, 22), (16, 6), (22, 6), (3, 1).

5
16 22 3 6 1

Indicații: De pe prima linie a fișierului citim valoarea lui n , apoi, într-un ciclu, citim numerele de pe a doua linie în vectorul v (contorul i al ciclului va lua succesiv valorile $0, 1, \dots, n-1$, și la fiecare pas citim un număr din fișier în elementul $a[i]$ al vectorului). Odată construit vectorul, trebuie să formăm toate perechile de elemente de forma $(a[i], a[j])$. Pentru aceasta, este suficient să formăm toate perechile de indici de forma (i, j) , în două cicluri, cu $i=0, 1, \dots, n-2$ și $j=i+1, \dots, n-1$. Pentru fiecare astfel de pereche $(a[i], a[j])$, testăm dacă $a[i]$ și $a[j]$ sunt ambele pare SAU ambele impare, iar în caz afirmativ afișăm respectiva pereche.

► 24. Fișierul **SIR.TXT** conține pe primul rând un întreg n , iar pe al doilea rând un șir de n numere întregi separate prin spații. Realizați un program care determină perechile de elemente ale șirului (nu neapărat consecutive) cu proprietatea că ambele elemente ale perechii au aceeași sumă a cifrelor. Prechile

astfel determinate vor fi scrise în fișierul **PERECHI.TXT**, fiecare pereche pe câte un rând, cu cele două elemente ale sale separate prin spații.

Exemplu:

SIR.TXT

```
6
5 301 122 62 401 17
```

PERECHI.TXT

```
5 122
5 401
122 401
62 17
```

► **25. (Bacalaureat iulie 2009, varianta 22)**

Scrieți un program care citește din fișierul text **BAC.TXT** un numărul întreg n ($1 \leq n \leq 10000$) și un șir de n perechi de numere întregi (a, b) ($1 \leq a \leq b \leq 32000$), fiecare pereche fiind scrisă pe o linie nouă a fișierului, cu un spațiu între cele două numere. Programul afișează pe ecran pentru fiecare pereche (a, b) cel mai mare număr natural din intervalul închis $[a, b]$ care este o putere a lui 2 sau numărul 0 dacă nu există nicio putere a lui 2 în intervalul respectiv. Un număr p este putere a lui 2 dacă există un număr natural k astfel încât $p=2^k$.

Exemplu: dacă fișierul **BAC.TXT** conține numerele din figură, atunci se va afișa: 64 16 0.

```
3
2 69
10 20
19 25
```

► **26.** Fișierul text **TRIUNGHI.TXT** conține pe fiecare linie câte trei numere separate prin câte un spațiu. Nu se cunoaște câte astfel de triplete sunt în fișier. Pentru fiecare dintre aceste triplete să se verifice dacă pot reprezenta laturile unui triunghi (sunt pozitive și suma a oricare două este mai mare decât al treilea număr); în caz afirmativ se va stabili natura triunghiului în funcție de laturi. Pe ecran se va afișa pentru fiecare triplet, pe un rând, câte un mesaj sugestiv de genul: "ISOSCEL", "ECHILATERAL", "OARECARE", "NU E TRIUNGHI".

Exemplu: dacă fișierul are conținutul din figură, atunci pe ecran se vor tipări următoarele rânduri:

```
OARECARE
ISOSCEL
ISOSCEL
NU E TRIUNGHI
ECHILATERAL
```

```
3 4 6
4 4 5
7 7 8
1 2 9
3 3 3
```

► **27. (Bacalaureat iulie 2009, varianta 31)**

În fișierul **NUMERE.TXT** se află memorate, pe prima linie un număr natural n ($1 \leq n \leq 100$), iar pe fiecare dintre următoarele n linii, câte două numere întregi x, y ($-100 \leq x \leq y \leq 100$), reprezentând capetele câte unui segment $[x, y]$ desenat pe axa Ox a sistemului de coordonate carteziane.

a) Scrieți în limbajul C/C++ un program eficient din punct de vedere al timpului de executare și al spațiului de memorare, care citește din fișier datele

existente, determină segmentul rezultat în urma intersecției tuturor celor n segmente date și afișează pe ecran două numere despărțite printr-un spațiu ce reprezintă capetele segmentului cerut. Dacă segmentele nu au niciun punct comun se va afișa pe ecran valoarea 0.

b) Descrieți în limbaj natural algoritmul utilizat, justificând eficiența acestuia.

Exemplu: dacă fișierul **NUMERE.TXT** are conținutul din imagine, se va afișa pe ecran: 3 5

```
5
-7 10
3 20
-5 5
0 12
-8 30
```

► **28.** Să se rezolve ecuațiile de gradul al doilea ale căror coeficienți se găsesc în fișierul **COEF.TXT**. Fișierul conține pe fiecare linie trei numere reale, separate prin spații, reprezentând coeficienții unei ecuații, și nu se cunoaște numărul liniilor. Pentru fiecare ecuație, se va scrie pe un rând al ecranului: rădăcinile ecuației separate prin spații dacă ecuația este rezolvabilă, respectiv mesajul "nu are soluții" în caz contrar.

Exemplu: dacă fișierul are conținutul din imagine, atunci pe ecran se vor afișa următoarele rânduri:

```
1 2
1 1
nu are solutii
2 -1
```

```
1 -3 2
1 -2 1
1 1 -3
1 -1 -2
```

► **29. (Bacalaureat iulie 2009, varianta 31)**

Fișierul text **NUMERE.IN** conține, pe fiecare linie, câte două numere naturale nenule mai mici sau egale decât 10000, despărțite printr-un spațiu, reprezentând numitorul și numărătorul câte unei fracții. Scrieți un program care, pentru fiecare linie k din fișierul **NUMERE.IN**, citește numitorul și numărătorul fracției de pe această linie și scrie în fișierul text **NUMERE.OUT**, tot pe linia k , numitorul și numărătorul acestei fracții, adusă la forma ireductibilă, ca în exemplu.

Exemplu: dacă fișierul **NUMERE.IN** are conținutul alăturat:

```
12 14
11 12
2 2
4 8
```

atunci fișierul **NUMERE.OUT** va avea următorul conținut:

```
6 7
11 12
1 1
1 2
```

► **30.** În fișierul **NR.TXT** se găsesc n perechi de numere întregi. Fișierul conține: pe primul rând valoarea lui n , apoi, pe fiecare din următoarele n rânduri, elementele unei perechi separate printr-un spațiu. Să se afișeze numărul perechilor cu proprietatea că media aritmetică a celor două numere din pereche este egală cu o valoare dată M (unde M se citește de la tastatură).

Exemplu: pentru $M=2$ și fișierul din figură, programul va afișa valoarea 3.

```
1 3
1 2
2 2
3 9
-2 6
0 1
```

► **31.** În fișierul **SIR.IN** se găsește un șir de numere întregi, scrise toate pe un rând, separate prin câte un spațiu. Nu se cunoaște numărul de elemente ale șirului. Realizați un program care inserează între oricare două elemente ale șirului media lor aritmetică. Elementele șirului rezultat după inserare, se vor scrie în fișierul **SIR.OUT**, unul sub altul (câte unul pe fiecare rând).

Indicații: A insera câte o valoare între oricare două elemente, înseamnă a face o inserare înaintea fiecărui element începând cu al doilea până la ultimul. Altfel spus, într-un ciclu, la fiecare pas:

– inserăm valoarea $m = (v[i-1] + v[i]) / 2$, înaintea elementului $v[i]$, adică pe poziția i ; în acest scop, vom muta cu o poziție mai la dreapta toate elementele de la $v[i]$ inclusiv până la sfârșitul vectorului, adică într-un alt ciclu cu j de la i la $n-1$, facem $v[j+1] = v[j]$. Apoi, aducem valoarea dorită m pe poziția i , prin simpla atribuire $v[i] = m$.

– facem $i = i + 1$, pentru a merge la următorul element al vectorului inițial, adică pentru a trece peste valoarea tocmai adăugată în vector.

► **32. (Bacalaureat iulie 2005, enunț modificat)**

Fișierul text **BAC.TXT** conține pe prima sa linie un șir de maxim 1000 de caractere, care pot fi litere cifre și spații. Realizați programul Pascal care afișează pe ecran câte caractere cifră există între al p -ulea și al q -ulea caracter din fișier (unde valorile pozițiilor p și q se citesc de la tastatură).

Exemplu: dacă $p=2$, $q=8$, iar fișierul **BAC.TXT** conține succesiunea de caractere **mx12a3pb55...**, unde cifra 5 se repetă de 961 ori, atunci programul va afișa valoarea 3, deoarece între pozițiile 2 și 8 din fișier se găsesc trei caractere cifră.

► **33. (Bacalaureat iulie 2009, varianta 31)**

Se citește de pe prima linie a fișierului text **NUMERE.IN** un număr natural n ($0 < n < 10000$) iar de pe a doua linie a fișierului n numere naturale din intervalul $[1, 100]$ și se cere să se afișeze pe ecran, despărțite prin câte un spațiu, numărul sau numerele întregi din intervalul $[1, 100]$ care nu apar printre numerele citite. Dacă pe a doua linie a fișierului apar toate numerele din intervalul precizat, se va afișa mesajul **NU LIPSESTE NICIUN NUMAR**. Alegeți un algoritm de rezolvare eficient din punctul de vedere al timpului de executare.

Exemplu: pentru fișierul **NUMERE.IN** cu următorul conținut:

```
12
4 2 3 1 6 5 7 8 9 11 10 100
```

se vor afișa valorile 12 13 ... 99.

a) Explicați în limbaj natural metoda utilizată justificând eficiența acesteia (4-6 rânduri)

b) Scrieți programul ce rezolvă problema enunțată, corespunzător metodei descrise la punctul a).

Indicații: O metodă eficientă constă în folosirea unui vector v , cu poziții de la 1 la 100, în care fiecare element $v[i]$ va memora numărul de apariții ale valorii i în fișier. Mai întâi inițializăm tot vectorul v cu 0, apoi, pentru completarea vectorului, citim succesiv într-un ciclu cele n numere din fișier, în aceeași variabilă x ; pentru fiecare x citit, incrementăm cu 1 elementul $v[x]$ care ne indică numărul de apariții ale lui x în fișier (am mai consemnat o apariție în plus față de câte erau în acel moment. În final,

programul va trebui să parcurgă vectorul, și pentru fiecare element $v[i]$ egal cu 0 (cu $i=1, 2, \dots, 100$), să afișeze poziția i .

► **34.** Se dau două fișiere text. Primul dintre acestea, numit **LITERE.TXT**, conține o mulțime de litere mari distincte ale alfabetului englez, scrise pe un singur rând, despărțite prin câte un spațiu. Al doilea fișier, numit **FRAZA.TXT** conține mai multe rânduri de text, scris numai cu litere mari, rânduri de lungimi diferite. Nu se cunoaște nici numărul caracterelor din primul fișier, nici numărul rândurilor din cel de-al doilea. Realizați un program care stabilește câte din literele existente în fișierul **LITERE.TXT** apar în fișierul **FRAZA.TXT** de cel puțin p ori (unde valoarea lui p se citește de la tastatură).

Exemplu: pentru $p=4$ și fișierele:

FRAZA.TXT

MIE IMI PLACE MULT
INFORMATICA

LITERE.TXT

M I U S

programul va afișa valoarea 2, întrucât două dintre literele aflate în fișierul **LITERE.TXT**, în speță 'M' și 'I', apar în fișierul **FRAZA.TXT** de cel puțin patru ori.

► **35. (Bacalaureat iulie 2000, varianta 7)**

Fișierul **BAC2000.TXT** este format din două linii. Prima linie este formată dintr-un șir de cel puțin una și cel mult 100 de litere mari nedespărțite prin spații, pe linia a doua aflându-se cel puțin unul și cel mult 100 de numere despărțite prin spații. Numerele de pe a doua linie sunt valori întregi din intervalul $[-1000, 1000]$. Conținutul fișierului respectă întocmai precizările anterioare, nefiind necesare validări.

Scrieți un program care să decidă dacă în fișier există tot atâtea litere pe prima linie câte numere se găsesc pe a doua linie. Se va afișa pe ecran mesajul "Da", dacă sunt tot atâtea litere pe prima linie câte numere sunt pe a doua linie, sau mesajul "Nu" în caz contrar.

► **36. (Bacalaureat iulie 2006)**

Scrieți un program care citește din fișierul **BAC.TXT** un șir S de maxim 100000 de numere naturale alcătuite din exact două cifre fiecare, și afișează distanța maximă care există între două elemente egale ale șirului. Definim distanța dintre două elemente s_i și s_j , ca fiind modulul diferenței indicilor celor două elemente, adică expresia $|i - j|$. Dacă șirul conține numai elemente distincte, distanța cerută este 0. Se cere o metodă eficientă de rezolvare.

Exemplu: dacă pe prima linie a fișierului se află șirul

14 12 10 10 14 15 10 12 90

atunci se afișează valoarea 6, deoarece distanța maximă dintre două elemente egale este 6 ($s_2 = s_8$ și $8 - 2 = 6$).

► **37. (Bacalaureat iulie 2009, varianta 91)**

Fișierul **NUMERE.IN** conține cel mult 5000 de numere reale, câte unul pe fiecare linie. Se cere să se scrie un program care să citească toate numerele din fișier și să afișeze pe ecran numărul de ordine al primei, respectiv al ultimei linii pe care se află cel mai mare număr din fișier. Cele două valori vor fi separate printr-un spațiu. Alegeți o metodă de rezolvare eficientă din punct de vedere al spațiului de memorare și al timpului de executare.

Exemplu: dacă fișierul are conținutul alăturat, pe ecran se vor afișa numerele 2 6.

a) Descrieți succint, în limbaj natural, metoda de rezolvare aleasă, explicând în ce constă eficiența ei.

b) Scrieți programul corespunzător metodei descrise.

3.5
7
-4
7
2
7
6.3
5

► 38. (Bacalaureat iulie 2009, varianta 10)

a) Scrieți un program care citește de la tastatură un număr natural n ($0 < n \leq 1000000$) și apoi n numere naturale nenule (cu cel mult șapte cifre fiecare) ordonate crescător și, utilizând un algoritm eficient din punct de vedere al memoriei utilizate și al timpului de executare, determină pentru fiecare număr citit cea mai mare valoare mai mică sau egală cu acesta ce reprezintă o putere a lui 3. Un număr natural y este putere a lui 3 dacă există un număr natural k astfel încât $y = 3^k$. Numerele astfel determinate vor fi scrise în fișierul **BAC.TXT**, separate prin câte un spațiu.

Exemplu: dacă $n=5$ și cele 5 numere citite au valorile 4 5 9 12 28 fișierul **BAC.TXT** va avea conținutul: 3 3 9 9 27

b) Descrieți succint, în limbaj natural, algoritmul pe baza căruia a fost scris programul de la punctul a), explicând în ce constă eficiența metodei folosite.

Indicații: citim separat primul număr din fișier într-o variabilă x și inițializăm cu 2 o variabilă t . Apoi, în mod repetat, înmulțim pe t cu 3 prin atribuirea $t := 3 * t$, până când $t > x$. Valoarea la care a ajuns t după acest ciclu este puterea căutată pentru primul număr din fișier. În continuare, pentru al doilea număr x , puterea cerută va fi cu siguranță mai mare decât cea întâlnită la primul număr! În consecință, vom citi succesiv celelalte n elemente ale șirului din fișier în aceeași variabilă x , și, pentru fiecare x citit, reluăm algoritmul înmulțirilor cu 3, dar cu un amendament: la fiecare nou x , variabila t nu va mai pleca de la 3, ci de la valoarea cu care a rămas la finele căutării pentru x -ul anterior.



Capitolul II: Principiile programării structurate

II.3.1. Structura de selecție simplă

1. valoarea 1 2. $a=4, b=3, c=2$ 3. $(a>b)$ SAU $(a>c)$ 4. $a=1$ și $b=0$ 5. b) 6. c) 7. a)

II.4.1. Structura repetitivă cu test inițial

1. două valori, și anume $n=5$ și $n=6$; 2. c) 3. c) 4. c) 5. 38, 47, 56
6. orice numere care se termină în cifra 1 respectiv 9 (221 și 149, sau 81 și 39, etc.)

II.4.2. Structura repetitivă cu test final

1. valoarea 16 2. numărul 33 3. valoarea 135 4. $x=4$ și $y=6$ (cu rezultat $p=24$);
5. c)

II.4.3. Structura repetitivă cu număr fix de pași

1. a) 2. valoarea 6 3. valoarea 6 4. toate cifrele impare, adică 1,3,5,7,9. 5. numărul 7
(pentru $i=4$ și $j=3$). 6. 28 de numere ($1+2+\dots+7=(7*8)/2$)

Capitolul III: Elementele de bază ale limbajului

III.1.3. Identificatori, separatori și comentarii

1. b) 2. c) 3. d)

III.2.3. Operatori

1. c) 2. d) 3. a) 32767 b) or c) relaționali d) #define 4. c) 5. b) 6. b) 7. b) 8. c) 9. c) 10. a)

III.2.4. Expresii

1. a) 2. a) 3. a) 4. c) 5. d) 6. valoarea 64 7. c) 8. valoarea 1 9. a) 24 b) 0 10. -42 11. b)
12. $(a*b+(a-b)/2)/((a-2)/(b-2)-a/b)$ 13. d) 14. a) 15. d) 16. b) 17. c)

III.2.5. Atribuirea în C++

1. b) 2. b) 3. c) 4. a) 5. c) 6. d) 7. c) 8. d) 9. b) 10. d) 11. d)

III.3.1. Afișarea datelor fără formatare:

1. d) 2. d) 3. valoarea 15 4. d) 5. d) 6. a) 7. b) 8. a)

III.4.2. Împărțirea cu rest. Operatorul "%"

1. c) 2. valoarea 2 3. c) 4. c) 5. d) 6. c) 7. c) 8. c) 9. d) 10. d) 11. b) 12. d) 13. d) 14. b)

III.4.3. Funcții predefinite pentru tipurile numerice

1. c) 2. b) 3. c) 4. a) 5. valoarea 9 6. c) 7. b) 8. a) 9. d) 10. b)

III.4.4. Tipuri ne-numerice

1. c) 2. c) 3. b) 4. b) 5. d)

Cap. IV. Instrucțiunile limbajului

IV.1.1 Instrucțiunea de selecție simplă

1. d) 2. da, pentru orice valori negative egale, de exemplu $a=-1$ și $b=-1$
3. numărul 5 4. toate sunt echivalente cu cea inițială 5. c) 6. b) 7. b) 8. de două ori 9. o singură eroare: prezența caracterului ";" după condiția " $x<-3$ " 10. c
11. o singură eroare: prezența caracterului ";" după condiția " $x<-3$ "
12. c) 13. d) 14. b)

IV.1.2. Instrucțiunea de selecție multiplă

1. b) 2. c) 3. c)

IV.2.1. Ciclu cu test inițial

1. b) 2. $x=7, y=5$; 3. a) 4. d) 5. e) 6. c) 7. a) 8. valoarea 1101
9. 75, 50, etc, orice număr divizibil de două ori cu 5
10. if ($n\%5$) $q=q*10+1$; else $q*=10$; 11. unsigned int n,q;
12. valoarea 5 13. c) 14. b) 15. c) 16. d) 17. c) 18. d) 19. valoarea 4

IV.2.2. Ciclu cu test final

1. d) 2. $x=6, y=40$; 3. $a=720$ și $b=6$; 4. $a=0$ și $b=30$;
5. imposibil, din cauza încheierii premature a ciclului. 6. d) 7. 13277231 și 1
8. orice număr în care ultima cifră se mai repetă de încă două ori, ca de exemplu $n=18858$
9. nu 10. sunt în total 14 atribuiri 11. imposibil 12. h) 13. a) 14. d) 15. c) 16. d)
17. valoarea -2 18. a)

IV.2.3. Ciclu cu contor

- 1.a) 2.d) 3.d) 4.a) 5.b) 6.c) 7.c) 8.c) 9.a) 10.d) 11.b) 12.b) 13.b) 14.a)

Cap. V. Tablouri

V.1.1. Noțiuni introductive

1. Afirmatiile adevărate sunt a), c) și e). 2. $v[0]:=2$; $v[2]:=4$;
4. Atribuiri corecte sunt a), d), e) și g)
5. 5.1.) $R:V[n/2]=0$; 5.2.) c) 6. b) 7. b)

V.1.3. Algoritmi de prelucrare a vectorilor

1. valoarea 21 2. d) 3. valoarea 21 4. if ($i\%2$) $S+=v[i]$;
5. pentru ($i <= n-1, \dots, 1, 0$) execute 7. $2n+1$ atribuiri
8. $v=(0, 1, 4, 9)$ 9. a) 10. d) 11. (1,2,3,4,5,6,7,9,10)
12. 55 de interschimbări (adică $10+9+\dots+1+0$) 13. b) 14. b) 15. c) 16. b)

VI.1. Fișiere text

1. a) 2. a) 3. d) 4. b) 5. b) 6. c)

Cuprins

Capitolul I Algoritmi	3
I.1. Generalități despre algoritmi	3
I.1.1. Noțiunea de algoritm	3
I.1.2. Obiectele cu care lucrează algoritmi	4
I.2. Reprezentarea algoritmilor. Pseudocodul	7
I.2.1. Primul exemplu de pseudocod	8
Capitolul II Principiile programării structurate	12
II.1. Noțiunea de structură	12
II.2. Structura liniară	12
II.3. Structura alternativă (de selecție)	13
II.3.1. Structura de selecție simplă	13
Aplicație. R.II.1. Aria unui triunghi	15
Încercați singuri ! Tema nr. T.1.	15
Aplicație. R.II.2. Interpretarea unui pseudocod dat	16
Încercați singuri ! Tema nr. T.2.	18
Aplicație. R.II.3. Funcție matematică	20
Încercați singuri ! Tema nr. T.3.	20
Aplicație. R.II.4. Câtul și restul	21
Încercați singuri ! Tema nr. T.4.	22
Aplicație. R.II.5. Laturi în triunghi	22
Încercați singuri ! Tema nr. T.5.	23
Aplicație. R.II.6. Rezolvarea ecuației de gradul II	23
Încercați singuri ! Tema nr. T.6.	24
Ați înțeles ? Probleme cu răspuns scurt	25
Probleme propuse Tema nr. T.R.1.	26
II.3.2. Structura de selecție multiplă	27
Aplicație. R.II.7. Alegere	29
Încercați singuri ! Tema nr. T.7.	30
Aprofundare - Probleme propuse. Tema nr. T.R.2.	30
II.4. Structuri repetitive	31
II.4.1. Structura repetitivă cu test inițial	31
Aplicație. R.II.8. Produs de numere naturale impare	35
Încercați singuri ! Tema nr. T.8.	36
Aplicație. R.II.9. Numărare într-un șir de numere a cărui citire se termină cu 0	37
Încercați singuri ! Tema nr. T.9.	38
Aplicație. R.II.10. Maxim într-un șir de numere a cărui citire se termină cu 0	38
Încercați singuri ! Tema nr. T.10.	39
Aplicație. R.II.11. Suma cifrelor unui număr	40
Încercați singuri ! Tema nr. T.11.	41
Aplicație. R.II.12. Palindroame mai mici sau egale cu n	42
Încercați singuri ! Tema nr. T.12.	43
Aplicație. R.II.13. Pseudocod cu șir de numere	43
Încercați singuri ! Tema nr. T.13.	45
Aplicație. R.II.14. Pseudocod cu cifrele unui număr	47
Încercați singuri ! Tema nr. T.14.	49
Ați înțeles ? Probleme cu răspuns scurt	50
Aplicație. R.II.15. Frație zecimală	52
Încercați singuri ! Tema nr. T.15.	54
Aplicație. R.II.16. Număr prim	55
Încercați singuri ! Tema nr. T.16.	58
II.4.2. Structura repetitivă cu test final	58
Aplicație. R.II.17. Produs de numere naturale impare	60
Aplicație. R.II.18. Numărare într-un șir de numere a cărui citire se termină cu 0	61
Aplicație. R.II.19. Palindroame mai mici sau egale cu n	63
Aplicație. R.II.20. Interpretarea unui pseudocod dat	64
Încercați singuri ! Tema nr. T.20.	67
Aplicație. R.II.21. Multiplii comuni	70
Încercați singuri ! Tema nr. T.21.	71
Ați înțeles ? Probleme cu răspuns scurt	71
II.4.3. Structura repetitivă cu număr fix de pași	72
Aplicație. R.II.22. Număr prim	75

Aplicație. R.II.23. Interpretarea unui pseudocod dat.....	76
Încercați singuri ! Tema nr. T.24.....	79
Aplicație. R.II.24. Numere pare și divizibile cu 5.....	80
Încercați singuri ! Tema nr. T.24.....	84
Aprofundare – R.II.25. Șirul lui Fibonacci.....	81
Încercați singuri ! Tema nr. T.25.....	82
Aplicație. R.II. 26. Triunghi de numere.....	82
Încercați singuri ! Tema nr. T.26.....	83
Ați înțeles ? Probleme cu răspuns scurt.....	84
Probleme propuse Tema nr. T.R.3.....	85
Capitolul III Elemente de bază ale limbajului C++.....	93
III.1. Noțiuni introductive.....	93
III.1.1. Structura unui program C++.....	93
III.1.2. Vocabularul limbajului și setul de caractere.....	98
III.1.3. Identificatori, separatori și comentarii.....	98
Identificatori.....	98
Separatori și comentarii.....	99
Instrucțiunea vidă.....	99
Aplicație.R.III.1 Program scris corect.....	100
Ați înțeles ? Probleme cu răspuns scurt.....	100
III.2. Constante, variabile, operatori, expresii.....	101
III.2.1. Constante.....	101
Aplicație. R.III.2. Corespondențe.....	104
III.2.2. Variabile.....	105
III.2.3. Operatori.....	107
Ați înțeles ? Probleme cu răspuns scurt.....	107
III.2.4. Expresii.....	108
Aplicație. R.III.3.Expresie cu numere.....	111
Aplicație. R.III.4.Expresie aritmetică.....	111
Aplicație. R.III.5.Expresie logică.....	111
Ați înțeles ? Probleme cu răspuns scurt.....	112
III.2.5. Atribuirea în C++.....	114
Aplicație. R.III.6. Atribuiți.....	121
Ați înțeles ? Probleme cu răspuns scurt.....	122
III.3. Citirea și afișarea datelor.....	123
III.3.1. Afișarea datelor fără formatare.....	123
Ați înțeles ? Probleme cu răspuns scurt.....	127
III.3.2. Citirea datelor fără formatare.....	129
Aplicație. R.III.7. Măsura unui unghi.....	130
Aplicație. R.III.8. Interschimbarea a două numere.....	131
Încercați singuri !.....	131
III.4. Tipurile de date predefinite ale limbajului.....	132
III.4.1. Tipurile numerice.....	132
III.4.2. Împărțirea cu rest. Operatorul %.....	133
Aplicație. R.III.9. Expresii cu / și %.....	134
Ați înțeles ? Probleme cu răspuns scurt.....	135
III.4.3. Funcții predefinite pentru tipurile numerice.....	137
Aplicație. R.III.10. Valoarea unei expresii.....	145
Ați înțeles ? Probleme cu răspuns scurt.....	139
III.4.4. Tipuri ne-numerice.....	140
Ați înțeles ? Probleme cu răspuns scurt.....	142
III.5. Instrucțiuni pentru lucrul cu ecranul.....	144
Extindere: Expresii condiționale.....	145
Operatorul sizeof.....	146
Operatorul virgulă.....	147
Probleme propuse.....	149
Capitolul IV Instrucțiunile limbajului.....	151
IV.1. Instrucțiuni de selecție.....	151
IV.1.1. Instrucțiunea de selecție simplă.....	151
Aplicație. R.IV.1. Aria unui triunghi.....	152
Încercați singuri !.....	153
Aplicație. R.IV.2. Funcție matematică.....	154
Încercați singuri !.....	155
Aplicație. R.IV.3. Problemă cu răspuns scurt.....	155

Aplicație. R.IV.4. Câtul și restul.....	156
Încercați singuri !.....	157
Aplicație. R.IV.5. Latură în triunghi.....	157
Încercați singuri !.....	158
Aplicație. R.IV.6. Rezolvarea ecuației de gradul II.....	158
Încercați singuri.....	159
Ați înțeles ? Probleme cu răspuns scurt.....	160
Aprofundare - Probleme propuse.....	163
IV.1.2. Instrucțiunea de selecție multiplă.....	163
Aplicație. R.IV.7. Grilă.....	165
Aplicație. R.IV.8. Alegere.....	166
Încercați singuri !.....	166
Ați înțeles ? Probleme cu răspuns scurt.....	167
Aprofundare - Probleme propuse.....	167
IV.2. Instrucțiuni repetitive (cicluri).....	168
IV.2.1. Ciclul cu test inițial (while-do).....	168
Aplicație. R.IV.9. Problemă cu răspuns scurt.....	168
Aplicație. R.IV.10. Produs de numere naturale impare.....	169
Încercați singuri !.....	170
Aplicație. R.IV.11. Numărare într-un șir de numere a cărui citire se termină cu 0.....	171
Încercați singuri !.....	171
Aplicație. R.IV.12. Maxim într-un șir de numere a cărui citire se termină cu 0.....	172
Încercați singuri !.....	173
Aplicație. R.IV.13. Suma cifrelor unui număr.....	173
Încercați singuri !.....	174
Aplicație. R.IV.14. Frație zecimală.....	174
Încercați singuri !.....	176
Aplicație. R.IV.15. Număr prim.....	192
Încercați singuri !.....	178
Ați înțeles ? Probleme cu răspuns scurt.....	178
Probleme propuse.....	182
IV.2.2. Ciclul cu test final (do-while).....	182
Aplicație. R.IV.16. Afișarea unui șir de litere.....	183
Aplicație. R.IV.17. Produs de numere pare.....	184
Încercați singuri !.....	184
Aplicație. R.IV.18. Șir de numere a cărui citire se termină cu 0.....	184
Încercați singuri !.....	185
Aplicație. R.IV.19. Suma cifrelor unui număr natural.....	185
Încercați singuri !.....	186
Aplicație. R.IV.20. Multiplii comuni.....	186
Încercați singuri !.....	188
Ați înțeles ? Probleme cu răspuns scurt.....	188
Probleme propuse.....	191
IV.2.3. Ciclul cu număr fix de pași (for).....	191
Aplicație. R.IV.21.Secvențe.....	214
Aplicație. R.IV.22. Număr prim.....	196
Încercați singuri !.....	197
Aplicație. R.IV.23. Numere pare și divizibile cu 5.....	197
Încercați singuri !.....	198
Aplicație. R.IV.24. Șirul lui Fibonacci.....	198
Încercați singuri !.....	199
Aplicație. R.IV.25. Triunghi de numere.....	199
Încercați singuri !.....	200
Ați înțeles ? Probleme cu răspuns scurt.....	200
Capitolul V: Tablouri.....	204
V.1. Vectori (tablouri unidimensionale).....	204
V.1.1. Noțiuni introductive.....	204
Declararea unui vector.....	205
Numărul real (efectiv) de elemente.....	206
Inițializarea unui vector la declarare.....	207
Aplicație. R.V.1. Declarații de vector.....	208
Ați înțeles ? Probleme cu răspuns scurt.....	209
V.1.2. Parcurgerea, citirea și afișarea unui vector.....	210
Aplicație. R.V.2. Valori distincte nenule.....	213

Aplicație. R.V.3. Medie aritmetică într-un șir	214
Încercați singuri !	216
Aplicație. R.V.4. Elementele prime ale unui șir.....	217
Încercați singuri	218
V.1.3. Algoritmi fundamentali de prelucrare a tablourilor.....	219
Algoritm Determinarea minimului dintr-un șir de numere	219
Algoritm Determinarea maximului dintr-un șir de numere	220
Aplicație. R.V.5. Interschimbarea între minim și maxim	220
Încercați singuri !	223
Algoritm Căutarea unei valori într-un vector	224
Aplicație. R.V.6. Intersecția a două mulțimi	225
Încercați singuri !	226
Aplicație. R.V.7. Perechi de elemente într-un șir.....	227
Încercați singuri !	229
Algoritm - Sortarea unui vector prin metoda interschimbărilor directe.....	229
Algoritm - Sortarea unui vector prin metoda bulelor.....	231
Algoritm - Interclasarea a doi vectori sortați.....	235
Încercați singuri	238
Ați înțeles – Probleme cu răspuns scurt.....	238
Aprofundare. Probleme propuse	241
Capitolul VI: Fișiere.....	250
VI.1. Fișiere text.....	250
VI.1.1. Noțiunea de fișier de date.....	250
VI.1.2. Clasificarea fișierelor de date.....	251
VI.1.3. Noțiunea de fișier text.....	284
VI.1.4. Noțiunea de descriptor de fișier	252
VI.1.5. Prelucrarea fișierelor text cu instrumente care folosesc obiecte	253
Aplicație. R.VI.1. Șir de numere citite dintr-un fișier.....	258
Încercați singuri !	259
Aplicație. R.VI.2. Intervale.....	260
Încercați singuri !	262
Aplicație. R.VI.3. Sortarea unui șir de numere citit din fișier	262
Încercați singuri !	264
Aplicație. R.VI.4. Numere din fișier cu prima cifră pară.....	265
Încercați singuri !	267
Aplicație. R.VI.5. Fișiere identice.....	267
Încercați singuri !	268
Probleme propuse.....	269
Răspunsuri la "Întrebări cu răspuns scurt"	306
Cuprins	306