EVALUARE:
- lucrare scrisă / proiecte din partea de Algoritmi (~ în săptămâna a 8-a, dar se poate reda la examenul din iarnă) => nota1
- lucrare scrisă din partea de Structuri de date (în sesiunea de iarnă) => nota2
- teme pentru acasă => nota_teme

Nota finală (calculată doar dacă nota1≥5 şi nota2≥5):
40%*nota1 + 40%*nota2 + 20%*nota_teme.

BIBLIOGRAFIE propusă (partea de Algoritmi):

1. Donald Knuth, *Arta programării calculatoarelor*", vol.1 *Algoritmi fundamentali*, Editura Teora, 1999 (traducere).
2. Donald Knuth, *Arta programării calculatoarelor*", vol.2 *Algoritmi seminumerici*, Editura Teora, 2000 (traducere).
3. T.H. Cormen, C.E. Leiserson, R.L. Rivest, *Introducere în algoritmi*, Editura Libris Agora, 2001 (traducere).
4. R. Andonie, I. Gabarcea, *Alg. fundamentali. O perspectiva C++*, Ed. Libris, 1995.
5. Doina Logofătu, *Algoritmi fundamentali în C++. Aplicaţii*, Editura Polirom, 2007.
6. Doina Logofătu, *Bazele programării în C. Aplicaţii.,* Editura Polirom, 2006.
7. Marin Popa, Mariana Popa, *Elemente de algoritmi şi limbaje de programare*, Editura Universităţii din Bucureşti, 2005.
8. Octavian Pătrăşcoiu, Gheorghe Marian, Nicolae Mitroi, *Elemente de grafuri, combinatorică, metode, algoritmi şi programe*, Editura All, Bucureşti, 1994.
9. Gheorghe Barbu, Ion Văduva, Mircea Boloşteanu, *Bazele Informaticii*, Editura Tehnică, Bucureşti, 1997.
10. Viorel Păun, *Algoritmică şi programarea calculatoarelor. Limbajul C++.* Editura Universităţii din Piteşti, 2003.
11. Ghe. Barbu, Viorel Păun, *Calculatoare personale şi programare în C/C++*, Editura Didactică şi Pedagogică, Bucureşti, 2005.
12. **Ghe. Barbu, Viorel Păun, *Programarea în limbajul C/C++*, Editura MatrixRom, Bucureşti, 2011.**
13. **Iorga, V., Chiriţă, P., Stratan, C., Opincaru, C., *Programare în C/C++. Culegere de probleme*. Editura Niculescu, Bucureşti, 2003.**
14. B. Pătruţ, *Aplicaţii în C şi C++,* Editura Teora, 2000.
15. D.H. Logofătu, *C++. Probleme rezolvate şi algoritmi,* Editura Polirom, 2001.
16. **O. Catrina, Iuliana Cojocaru, *Turbo C++*, Editura Teora, 1993.**
17. V. Petrovici, F. Goicea, *Programare în limbajul C,* Editura Tehnică, 1993.
18. V. Cristea, C.Giumale, E. Kalisz, A. Pănoiu, *Limbajul C standard*, Ed. Teora, 1992.
19. C.Giumale, *Un atelier de programare*, Editura Computer Libris Agora, 2000.

ALGORITMI. METODE DE DESCRIERE A ALGORITMILOR (de citit)

1.1 Scurt istoric

În secolul al IX-lea d.Hr., un matematician persan, Abu Abdullah Muhammed bin Musa al-Khwarizmi a scris o lucrare despre **efectuarea calculelor numerice** într-o manieră algebrică, "Liber algorithmi", unde "algorithm" provine de la **al-Khwarizmi** ceea ce înseamnă "din oraşul Kwarizm", azi oraşul Kiwa din Uzbechistan. Acest autor, ca şi alţi matematicieni ai evului mediu înţelegeau prin algoritm o regulă pe bază căreia se pot efectua calcule matematice.

Multă vreme conceptul de algoritm rămâne cu o întrebuinţare

destul de restrânsă chiar şi în matematică. Către sfârşitul secolului al XIX-lea (1886-1888) Kronecker şi Dedekind introduc în matematică funcţiile recursive în care conceptul de algoritm este strâns legat de cel de recursivitate. Abia în secolul XX, în deceniile 3 şi 4, prin lucrările matematicienilor Skolem, Ackerman, Sudan, Gödel, Church, Kleene, Turing şi alţii, teoria algoritmilor şi recursivităţii se constituie ca atare.

Astăzi, ca rezultat al conexiunii dintre algoritm şi calculator, gândirea algoritmică s-a transformat dintr-un instrument matematic particular, într-o modalitate fundamentală de abordare a problemelor din diverse domenii, folosit pentru a descrie într-o manieră ordonată activităţi care constau în parcurgerea unei succesiuni de paşi (cum este de exemplu utilizarea unui telefon public sau realizarea unei reţete gastronomice).

Algoritmul este considerat noţiunea fundamentală a informaticii. **În informatică, prin algoritm se poate înţelege o metodă prin care se descriu paşii necesari pentru rezolvarea unei clase de probleme, metodă care se poate implementa pe calculator prin intermediul unui limbaj de programare.**

Definiţii alternative

În linii mari, despre un algoritm se pot afirma următoarele:

- Un algoritm este o mulţime de reguli ce se pot aplica în cadrul procesului de construcţie a soluţiei şi pot fi executate fi "de mână", fie de o maşină.
- Un algoritm este o secvenţă de paşi care transformă mulţimea datelor de intrare în datele de ieşire (rezultatele dorite).
- Un algoritm este o secvenţă de operaţii executate cu date ce trebuie organizate în structuri de date.
- Un algoritm este abstractizarea unui program care trebuie executat pe o maşină fizică (model de calcul).
- Un algoritm pentru o problemă dată este o mulţime de instrucţiuni care garantează găsirea unei soluţii corecte pentru orice instanţă a problemei, într-un număr finit de paşi.

1.2 Generalităţi despre algoritmi

În matematică există o serie de algoritmi: cel al rezolvării ecuaţiei de gradul doi, algoritmul lui Eratostene (pentru generarea numerelor prime mai mici decât o anumită valoare), schema lui Horner (pentru determinarea câtului şi restului împărţirii unui polinom la un binom), etc.

Soluţia problemei se obţine prin *execuţia* algoritmului. Algoritmul poate fi executat **pe o maşină formală** (în faza de proiectare şi analiză) sau **pe o maşină fizică** (calculator) după ce a fost codificat într-un **limbaj de programare**. Spre deosebire de un program, care depinde de un limbaj de programare, un algoritm este o entitate matematică care este independentă de maşina pe care va fi executat.

Studiul algoritmilor presupune:

- *Elaborarea algoritmilor*. Are ca scop identificarea unei soluţii de rezolvare a problemei practice.
- *Exprimarea algoritmilor*. Presupune prezentarea algoritmilor într-un limbaj abstract, bine definit, astfel încât să avem o formă clară şi concisă a soluţiei identificate. Algoritmii pot fi exprimaţi şi în limbaje de programare.
- *Validarea şi analiza algoritmilor*. Înseamnă revizuirea algoritmilor pentru a vedea dacă într-adevăr, acestia produc rezultatele dorite pentru problema analizată şi identificarea eficienţei acestor algoritmi
- *Testarea algoritmilor*. Presupune rularea programelor aferente algoritmilor şi depanarea acestora.

Un algoritm trebuie să posede următoarele ***proprietăţi***:

Finitudine. Un algoritm trebuie să admită o descriere finită şi fiecare dintre prelucrările pe care le conţine trebuie să poate fi executată în timp finit. Prin intermediul algoritmilor nu pot fi prelucrate structuri infinite.

Corectitudinea. Este proprietatea algoritmului de a furniza o soluţie corectă a problemei date.

Generalitate. Un algoritm destinat rezolvării unei (clase de) probleme trebuie să permită obţinerea rezultatului pentru orice date de intrare şi nu numai pentru date particulare de intrare.

Rigurozitate / claritate. Paşii algoritmului trebuie specificaţi riguros, fără ambiguităţi. În orice etapă a execuţiei algoritmului trebuie să se ştie exact care este următoarea etapă ce va fi executată.

Eficienţa. Algoritmii pot fi efectiv utilizaţi doar dacă folosesc *resurse de calcul* în volum acceptabil. Prin resurse de calcul se înţelege volumul de memorie şi timpul necesar pentru execuţie.

Exemple:

1. *Nu orice problemă poate fi rezolvată algoritmic.* Considerăm un număr natural n şi următoarele două probleme: (*i*) să se construiască mulţimea divizorilor lui n; (*ii*) să se construiască mulţimea multiplilor lui n. Pentru rezolvarea primei probleme se poate elabora uşor un algoritm, în schimb pentru a doua problemă nu se poate scrie un algoritm atâta timp cât nu se cunoaşte un criteriu de oprire a prelucrărilor.

2. *Un algoritm trebuie să funcţioneze pentru orice date de intrare.* Să considerăm problema ordonării crescătoare a şirului de valori: (2 1 4 3 5). O modalitate de ordonare ar fi următoarea: se compară primul element cu al doilea, iar dacă nu se află în ordinea bună se interschimbă, adică îşi schimbă locul (în felul acesta se obţine (1 2 4 3 5)); pentru şirul astfel transformat se compară al doilea element cu al treilea şi dacă nu se află în ordinea dorită se interschimbă (la această etapă şirul rămâne neschimbat); se continuă procedeul până penultimul element se compară cu ultimul. În felul acesta se obţine (1 2 3 4 5). Deşi metoda descrisă mai sus a permis ordonarea crescătoare a şirului (2 1 4 3 5) ea nu poate fi considerată un algoritm de sortare întrucât dacă este aplicată şirului (3 2 1 4 5) conduce la (2 1 3 4 5).

3. *Un algoritm trebuie să se oprească.* Se consideră următoarea secvenţă de prelucrări:

Pas 1. Atribuie variabilei x valoarea 1;

Pas 2. Măreşte valoarea lui x cu 2;

Pas 3. Dacă x este egal cu 100 atunci se opreşte prelucrarea, altfel se reia de la Pas 2.

Este uşor de observat că x nu va lua niciodată valoarea 100, deci succesiunea de prelucrări nu se termină niciodată. Din acest motiv nu poate fi considerat un algoritm corect.

4. *Prelucrările dintr-un algoritm trebuie să fie neambigue.* Considerăm următoarea secvenţă de prelucrări:

Pas 1. Atribuie variabilei x valoarea 0;

Pas 2. *Fie* se măreşte x cu 1 *fie* se micşorează x cu 1;

Pas 3. Dacă $-10 \leq x \leq 10$ se reia de la Pasul 2, altfel se opreşte algoritmul.

Atât timp cât nu se stabileşte un criteriu după care se decide dacă x se măreşte sau se micşorează secvenţa de mai sus nu poate fi considerată un algoritm. Ambiguitatea poate fi evitată prin utilizarea unui limbaj riguros de descriere a algoritmilor. Să considerăm că Pas 2 anterior se înlocuieşte cu:

Pas 2. Se aruncă o monedă. Dacă se obţine cap se măreşte x cu 1, iar dacă se obţine pajură se micşorează x cu 1;

În acest caz specificarea prelucrărilor nu mai este ambiguă chiar dacă la execuţii diferite se obţin rezultate diferite. Ce se poate spune despre finitudinea acestui algoritm? Dacă s-ar

obţine alternativ cap respectiv pajură, prelucrarea ar putea fi infinită. Există însă şi posibilitatea să se obţină de 11 ori la rând cap (sau pajură), caz în care procesul s-ar opri după 11 repetări ale pasului 2.
Atât timp cât şansa ca algoritmul să se termine este nenulă algoritmul poate fi considerat corect (în cazul prezentat mai sus este vorba despre un algoritm aleator).

5. *Un algoritm trebuie să se oprească după un interval rezonabil de timp.* Să considerăm că rezolvarea unei probleme implică prelucrarea a n date şi că numărul de prelucrări $T(n)$ depinde de n. Presupunem că timpul de execuţie a unei prelucrări este 10^{-3} s şi că problema are dimensiunea $n = 100$. Dacă se foloseşte un algoritm caracterizat prin $T(n) = n$ atunci timpul de execuţie va fi $100 \times 10^{-3} = 10^{-1}$ secunde. Dacă, însă se foloseşte un algoritm caracterizat prin $T(n) = 2^n$ atunci timpul de execuţie va fi de aproximativ 10^{19} ani.

Algoritmii pot fi descrişi (*reprezentaţi*) în mai multe moduri, folosind:
- *schema logică*, care constituie un mod sugestiv (intuitiv) de reprezentare a unui algoritm, utilizând simboluri cu o semnificaţie bine precizată;
- *pseudocod*, care se remarcă prin simplitatea şi naturaleţea sa, fiind de asemenea un limbaj cu reguli sintactice foarte simple ce permit exprimarea neambiguă a ordinii de execuţie a paşilor, etc.
- *un limbaj de programare* ca Pascal, C/C++, Java, etc.

Însă un algoritm poate fi reprezentat în orice limbaj, pornind de la limbajul natural până la limbajul de asamblare al unui calculator specific. De altfel, limbajele de programare pot fi considerate exemple de limbaje algoritmice.

1.3 Descrierea algoritmilor folosind pseudocodul

Pseudocodul reprezintă o notaţie textuală ce permite exprimarea logicii programelor într-un mod oarecum formalizat fără a fi necesare totuşi reguli de sintaxă riguroase ca în limbajele de programare. Nu există nici o definire formală sau unică a formei pe care trebuie să o aibă un pseudocod. În principiu, în orice pseudocod se folosesc două tipuri de propoziţii:
a) *propoziţii standard* prin care se exprimă operaţii ce pot fi transcrise direct într-un limbaj de programare, fiecare propoziţie începând cu un verb care exprimă cât mai fidel operaţia descrisă;
b) *propoziţii nestandard* prin care se exprimă operaţii ce urmează a fi detaliate ulterior, de exemplu:
   * se citesc datele de intrare
   * se afişează meniul pe ecran
   unde caracterul * este folosit pentru a marca începutul propoziţiilor de acest tip.

În continuare prezentăm un exemplu de pseudocod pe care-l vom folosi mai departe şi echivalentele în C/C++.
Operaţiile standard ale pseudocodului sunt descrise mai jos.

1. *Comanda de citire*. Citirea constă în transferul de valori de pe mediul de intrare în locaţiile de memorie ale calculatorului şi are sintaxa:

```
Pseudocod:  citeste lista_de_variabile
C:          int scanf(control, par1, par2, …);
C++:        cin>>var1>>var2…;
```

unde *control*: conţine texte şi specificatori de format, *par1, par2, ...* reprezintă adresele receptoare ale datelor (adrese de variabile), iar *var1, var2, ...* reprezintă variabilele în care se memorează datele de intrare.

2. *Comanda de scriere*. Scrierea sau afişarea constă în transferul valorii unor variabile din memorie pe mediul de ieşire, sau valoarea unor expresii şi are forma:

```
Pseudocod:   scrie lista_de_expresii
C:           int printf(control, par1, par2, …);
C++:         cout<<var1<<var2…;
```

unde *control* reprezintă un şir de caractere care conţine textul de afişat şi specificatori de format, *par1, par2, …* sunt expresiile, valorile care se scriu conform specificatorilor de format prezenti în parametrul de control.

3. *Comanda de atribuire*. Operaţia de atribuire constă în calculul unei expresii şi asocierea acestei valori unei variabile, conform sintaxei:

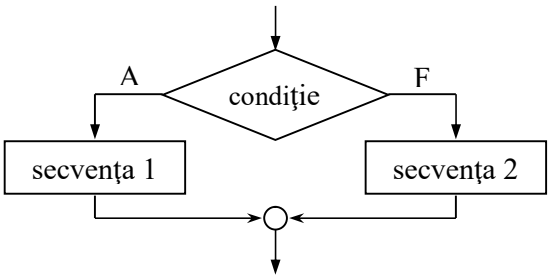
```
Pseudocod:   variabila ← expresie
C/C++:       variabila = expresie;
```

4. *Secvenţa*. Secvenţa desemnează un grup de instrucţiuni scrise una după alta, executate secvenţial şi delimitate grafic. Efectul execuţiei unei comenzi dintr-o secvenţă depinde de poziţia sa. Sintaxa acestor secvenţe este:

```
Pseudocod:
            instructiunea1
            instructiunea2
            ...
            instructiunean
C/C++:
            {
              instructiunea1
              instructiunea2
              ...
              instructiunean
            }
```

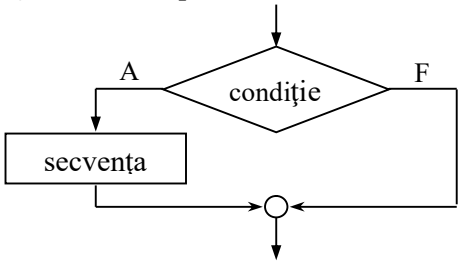
5. *Decizia* este o comanda ce specifică alegerea pentru execuţie a uneia dintre două alternative. Sintaxa comenzii este:

```
Pseudocod:
            daca conditie atunci
                  secventa1

            altfel
                  secventa2

C/C++:      if (conditie) secventa1;
            else secventa2;
```



unde `conditie` poate fi o variabilă logică, o expresie relaţională sau logică. Modul de execuţie al deciziei este următorul: se evaluează condiţia şi dacă este adevărată se execută $\texttt{secventa}_1$, în caz contrar se execută $\texttt{secventa}_2$.
În cazul în care cuvântul `altfel` lipseşte, se foloseşte forma simplificată:

```
Pseudocod:
            daca conditie atunci
                  secventa

C/C++:      if (conditie) secventa;
```



6. *Selecţia* reprezintă o extindere a operaţiei de decizie, permiţând alegerea unei variante din mai multe posibile. Forma generală este:

Pseudocod:

```
alege expresie dintre
      c1: secventa1
      c2: secventa2
      ..........
      cn: secventan
rest: secventan+1
```

C/C++:

```
switch (expresie)
{
 case c1:    secventa1; <break;>
 case c2:    secventa2; <break;>
 ...
 case cn:    secventan; <break;>
 <default:   secventan+1;>
}
```

unde c_1, c_2, …, c_n se numesc etichete şi se folosesc la identificarea secvenţelor. Eticheta `rest` şi $secventa_{n+1}$ sunt opţionale. Modul de execuţie este următorul:

- se evaluează expresia;
- se caută eticheta având valoarea egală cu valoarea expresiei şi este selectată secvenţa corespunzătoare;
- dacă nici o etichetă nu are valoarea expresiei, atunci este selectată $secventa_{n+1}$;
- se execută secvenţa selectată după care se trece la comanda următoare selecţiei.

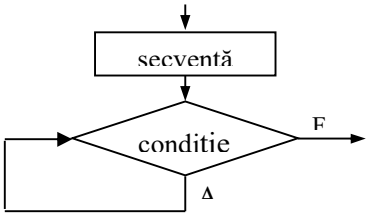
`Break` poate lipsi. În cazul în care acesta lipseşte, se vor executa toate instrucţiunile care urmează după selecţia corespunzătoare în jos. `Default` poate lipsi.

7. *Cicluri şi iteraţii*. În rezolvarea problemelor apare deseori situaţia în care o anumită secvenţă urmează a se relua de mai multe ori; o astfel de combinaţie se numeşte ciclu. Pentru reprezentarea ei se foloseşte o comandă de ciclare ce specifică atât grupul de instrucţiuni ce se repetă cât şi condiţia de repetare. Există trei tipuri de cicluri: ciclul cu test final, ciclul cu test iniţial şi ciclul cu contor.

7.1 *Ciclul cu test final*. Testul pentru repetarea calculelor se face după execuţia grupului de comenzi care trebuie repetate. În pseudocod, dacă condiţia nu este îndeplinită (falsă) se reia execuţia secvenţei, iar dacă condiţia este îndeplinită se trece la comanda următoare ciclului cu test final. În C, dacă condiţia este nenulă (adevărată) se reia execuţia secvenţei, iar dacă condiţia este nulă se reia ciclul "`do-while`". Sintaxa comenzii este:

Pseudocod:

```
repeta
    secventa
cat_timp conditie
```

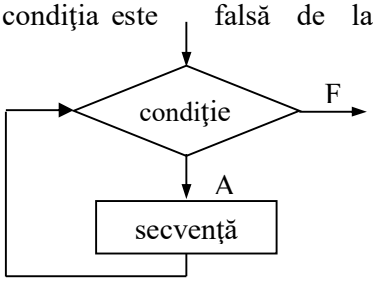


C/C++:

```
do
   secventa
while (conditie);
```

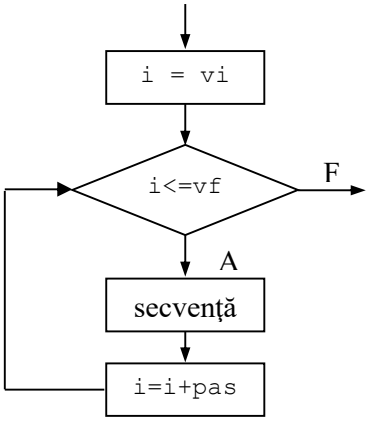
7.2 *Ciclul cu test iniţial*. Testul pentru reprezentarea calculelor se face înaintea execuţiei grupului de comenzi care trebuie repetate. Dacă condiţia este îndeplinită, se

execută secvenţa după care se reevaluează condiţia, iar dacă condiţia nu este îndeplinită, se trece la comanda următoare ciclului cu test iniţial. Acest tip de ciclu are avantajul că secvenţa nu se execută nici o dată dacă condiţia este falsă de la început. Sintaxa comenzii este:




```
Pseudocod:
    cat_timp conditie repeta
        secventa

C/C++: while (conditie)
         secventa;
```

7.3 *Ciclul cu contor*. Ciclul cu contor realizează repetarea unei secvenţe de un număr de ori prestabilit, controlat de o variabilă de ciclare numită contor. Sintaxa comenzii este:




```
Pseudocod:
    pentru i=vi,vf,pas repeta
        secventa

C/C++: for(i=vi;i<=vf;i+=pas) secventa;
```

unde:

- `i` este variabila de contorizare;
- `pas` este pasul cu care se face incrementarea;
- `vi`, `vf` sunt valorile initiale şi finale ale contorului.

Modul de execuţie este următorul:

1. se atribuie `i` ← `vi`;
2. se evaluează condiţia `i > vf`; dacă condiţia nu este îndeplinită se trece la pasul 3, altfel se trece la pasul 5;
3. se execută secvenţa;
4. se atribuie `i` ← `i + pas` şi se trece la pasul 2;
5. se execută comanda următoare ciclului cu contor.

Funcţionarea algoritmului a fost descrisă pentru valori pozitive ale lui `pas`; dacă `pas` este o valoare negativă, la pasul 2 se evaluează condiţia `i < vf`.

8. *Comanda de ieşire*. Aceasta comandă are rolul de a asigura terminarea forţată a unui ciclu. Sintaxa sa este:

```
Pseudocod:  iesire
C/C++:      exit(parametru);
```

9. *Comanda goto*. Aceasta comandă permite transferul explicit al execuţiei la o instrucţiune care are o anumită etichetă. Sintaxa sa este:

```
Pseudocod:  goto eticheta
C/C++:      goto (numar_eticheta);
```

10. *Proceduri şi funcţii*. În rezolvarea problemelor apar situaţii în care porţiuni de program similare se întâlnesc în diferite zone ale programului. Grupul de comenzi care se repetă poate constitui o unitate distinctă căreia i se dă un nume şi un set de parametri. Ori de câte ori va fi necesară execuţia acestui grup de comenzi se specifică numele şi parametrii care actualizează grupul de comenzi în noile condiţii. Grupul de comenzi se numeşte subprogram. Subprogramul poate fi procedură sau funcţie.

11.1 *Procedura*. Procedura specifică secvenţe de operaţii care se repetă în diferite puncte ale programului şi evidenţiază parametrii. Forma generală a procedurii în pseudocod este:

```
Pseudocod:
      procedura nume (lista_parametrilor_formali) este
            secventa
      sfarsit
C/C++: void nume(lista_parametrilor_formali)
       {
          secventa;
       }
```

Parametrii care apar în scrierea unei proceduri se numesc parametri formali, ei având un rol decriptiv. Apelul unei proceduri se face cu comanda:

```
Pseudocod: executa nume(lista_parametri_actuali)
C:     nume (lista_parametri_actuali);
```

11.2 *Funcţia*. Atunci când procedura calculează o singură valoare se poate utiliza un tip particular de procedură numită funcţie. Forma generală este:

```
Pseudocod:
      functie nume (lista_parametrilor_formali) este
            secventa
      sfarsit
C/C++: tip nume(lista_parametrilor_formali)
       {
          secventa;
          return valoare_tip;
       }
```

Pentru ca utilizarea unei funcţii în cadrul unei expresii să fie cât mai simplă, apelul unei funcţii este considerat ca un operand al expresiei. Forma apelului funcţiei este:

```
Pseudocod:   nume(lista_parametri_actuali)
C/C++:       variabila = nume(lista_parametri_actuali);
```

Execuţia unei funcţii intervine atunci când la evaluarea unei expresii este necesară valoarea sa. În această situaţie se face transmiterea parametrilor funcţiei şi se execută secvenţa de operaţii corespunzătoare, găsindu-se astfel valoarea funcţiei.

Utilizarea procedurilor şi funcţiilor se face în următoarele scopuri:

1. o secţiune de program utilizată în mai multe puncte ale acestuia poate fi descrisă o singură dată şi apelată ori de câte ori este nevoie, scurtându-se dimensiunea programului;
2. proiectarea unor părţi ale algoritmului se poate face separat, independent una de alta, putându-se codifica şi testa separat;
3. reprezentarea de sine stătătoare a rezolvării unor subprobleme;
4. construirea unor algoritmi de complexitate mare folosind deja module proiectate şi verificate.

Alte elemente de bază în pseudocod

Cuvinte cheie (keywords) sunt cuvintele rezervate pentru comenzile standard şi în curs le-am evidenţiat prin scrierea lor folosind tipul îngroşat. De exemplu: citeste, scrie, daca, atunci, altfel, alege, dintre, rest, repeta, pentru, etc.

Identificatori reprezintă un şir de caractere folosit pentru *nume de variabile*, *nume de funcţii*, etc. O restricţie de bază pentru identificatori este să difere de cuvintele cheie. În C/C++, un identificator începe întotdeauna cu o literă (a..z, A..Z) şi apoi se pot folosi cifre, litere şi caracterul '_' (underscore).

O **variabilă** este un identificator dat după dorinţă de către utilizator unei zone din memorie cu care se va lucra (stoca sau modifica datele utile programului) pe parcursul scrierii algoritmului/codului sursă. Numele unei variabile se alege de obicei astfel încât să descrie ce înseamnă pentru programator, din punct de vedere al pseudocodului într-o formă care poate fi prescurtată sau nu, după preferinţă. De exemplu, numele unei variabile ce reţine o notă poate fi: nota_1 sau n1 sau Nota_1, etc.

Caracteristicile variabilelor sunt:
- fiecare variabilă are un **nume**
- fiecare variabilă reţine **o valoare** sau un set de valori utile pentru algoritm
- fiecare variabilă va aparţine unui tip de dată (întregi, reale, caracter).

Tipul unei date determină valorile pe care le poate lua, dimensiunea zonei de memorie ocupate şi modul in care este reprezentată valoare în memorie.

Algoritmi cu ramificaţii

Definiţie: Algoritmii care pot fi descrişi folosind pe lângă comenzi/instrucţiuni de citire (în pseudocod: citeste), scriere (în pseudocod: scrie), atribuire (în pseudocod: ←) şi comenzi de decizie (în pseudocod: daca-atunci-altfel) sau selecţie (în pseudocod: alege) se numesc *algoritmi cu ramificaţii*.

Exemple de algoritmi cu ramificaţii:

R1 (1*).

Enunţul problemei: Să se scrie un algoritm pentru a determina greutatea ideală a unei persoane, cunoscând înălţimea h (în cm), vârsta v (în ani) şi sexul s (f pentru feminin sau m pentru masculin) şi folosind formulele: $G_m = 50 + 0{,}75(h-150) + (v-20)/4$, iar $G_f = G_m - 10$.

Metoda de rezolvare: Datele de intrare sunt h, v şi s. Apoi, indiferent de sexul persoanei, trebuie calculată valoarea expresiei $50 + 0{,}75(h-150) + (v-20)/4$ (în G_m), iar la final, dacă este persoană de sex feminin se mai scade 10 (din Gm) şi se afişează valoarea corespunzătoare.

Descrierea algoritmului în pseudocod:

```
citeste h, v, s
Gm ← 50 + 0,75(h-150) + (v-20)/4
daca s = f atunci
      Gf ← Gm - 10
      scrie Gf
altfel
      scrie Gm
```

Se mai poate folosi o singură variabilă pentru calculul greutăţii:

```
citeste h, v, s
G ← 50 + 0,75(h-150) + (v-20)/4  *pt o persoana de sex masculin
daca s = f atunci
      G ← G - 10 *daca este pers fem se scade 10 din vechea val
scrie G
```

Descrierea algoritmului în C++:

```
#include <iostream>
using namespace std;
int main() {
 int h,v;
 char s;
 float G;
 cout<<"Dati inaltimea persoanei (in cm): "; cin>>h;
 cout<<"Dati varsta persoanei (in ani): "; cin>>v;
 cout<<"Dati sexul persoanei (m sau f): "; cin>>s;
 G = 50 + 0.75*(h-150) + (v-20)/4.0; //greutatea unui barbat
 if (s =='f') G = G - 10; //sau G-=10; //greutatea unei femei
 cout << G << endl; //se afiseaza valoarea finala a var. G
 return 0;
}
```

După rularea programului C++, pe ecran va apare:

```
Dati inaltimea persoanei (in cm): 175 <Enter>
Dati varsta persoanei (in ani): 30 <Enter>
Dati sexul persoanei (s sau f): m <Enter>
70.75
```

Câteva explicaţii cod C++:

Înălţimea (pentru că este exprimată în cm) şi vârsta (fiind exprimată în ani) le-am declarat de tip întreg (am ales tipul `int`). Greutatea, însă este valoarea unei expresii care poate da cu virgulă şi atunci este declarată de tip real (am ales tipul `float`).

În variabila *G* se determină valoarea greutăţii ca pentru o persoană de sex masculin. Apoi, dacă este totuşi o persoană de sex feminin, se modifică valoarea acestei variabile, scăzând 10 din vechea valoare. La final, se afişează valoarea finală a acestei variabile.

R2 (3-4*).

Enunţul problemei: Descrieţi un algoritm pentru a determina diferenţa dintre două momente de timp ($T_1 \geq T_2$) exprimate în ore, min şi secunde (fără a folosi conversiile din ore-min-sec în secunde şi invers).

Metoda de rezolvare: Să luăm câteva exemple:

2h 40min 30sec –	2h 40min 15sec –	2h 30min 15sec –	2h 0min 15sec –
1h 30min 15sec	1h 30min 30sec	1h 40min 30sec	1h 30min 30sec
1h 10min 15sec	1h 9min 45sec	0h 49min 45sec	0h 29min 45sec

Un algoritm constă în compararea pe rând a numărului de secunde, respectiv minute de la cele două momente de timp (compararea orelor nu mai este necesară căci $T_1 \geq T_2$). Când nu se va putea face scăderea la secunde, se împrumută un minut de la T_1, iar când nu se va putea face scăderea la minute, se împrumută o oră de la T_1.

Mai jos, notăm cu h1, m1, s1 variabilele care memorează numărul de ore, minute, secunde de la primul moment de timp, cu h2, m2, s2 variabilele care memorează numărul de ore, minute, secunde de la al doilea moment de timp, respectiv cu h, m, s variabilele care memorează numărul de ore, minute, secunde de la moment de timp final.

Descrierea algoritmului în pseudocod:

```
citeşte h1,m1,s1,h2,m2,s2
daca s1 ≥ s2 atunci     *secundele se pot scadea
     s ← s1-s2
altfel      *s1<s2 => luam 1 min.
     daca m1>0 atunci       *avem 1 min. de luat
          m1 ← m1-1         *luam minutul = 60 sec.
          s = 60 + s1 - s2
     altfel *m1=0 => luam 1ora = 60min din care dam 1 min. la sec.
          h1 ← h1-1    *luam 1 ora = 60 min.
          m1 ← 60-1    *am primit 60 min. din care unul se da
          s = 60 + s1 - s2
daca m1 ≥ m2 atunci     *minutele se pot scadea
     m ← m1-m2
altfel      *m1<m2 => luam 1 oră (avem de unde pentru că T1 ≥ T2).
     h1 ← h1-1          *luam 1 oră = 60 min.
     m = 60 + m1 - m2
h ← h1-h2   *pentru că T1 ≥ T2 orele se scad fără probleme
scrie h,m,s
```

Descrierea algoritmului în C++ (în CodeBlocks):

```
#include <iostream>
using namespace std;
int main()
{
 int h1,m1,s1,h2,m2,s2,h,m,s;
 cout<<"Dati T1 (ore-min-sec): ";
 cin>>h1>>m1>>s1; //datele se despart prin spatiu sau Enter
 cout<<"Dati T2 (ore-min-sec): ";
 cin>>h2>>m2>>s2;
 if (s1>=s2) s = s1-s2; //secundele se pot scadea
 else            //s1>s2 => luam 1 min
    if (m1>0) //avem 1 min. de luat
    {
        m1--; //luam minutul
        s = 60+s1-s2; //1 min = 60 sec.
    }
    else //m1=0=>luam 1h=60min, din care dam 1min.la sec.
    {
        h1--; //luam 1h
        m1 = 60-1; //1h=60min din care dam 1 min
        s = 60+s1-s2;
    }
 if (m1>=m2) m = m1-m2;
 else
 {
    h1--;
    m = 60+m1-m2;
 }
 h = h1-h2;
 cout<<"T1-T2 = "<<h<<"h "<<m<<"min. "<<s<<"sec."<<endl;
 return 0;
}
```

Rulare:

```
Dati T1 (ore-min-sec): 2 30 15 <Enter>
Dati T1 (ore-min-sec): 1 40 30 <Enter>
T1-T2 = 0h 49min. 45sec.
```

Observaţie: În algoritmul de mai sus am urmărit modul în care se învaţă la şcoală scăderea a două momente de timp. Algoritmul însă se poate rezolva şi altfel. Una dintre optimizări este detaliată mai jos:

Descrierea algoritmului în pseudocod:

```
citeşte h1,m1,s1,h2,m2,s2
s ← s1-s2 *scădem secundele chiar dacă dă negativ
m ← m1-m2 *scădem minutele chiar dacă dă negativ
h ← h1-h *scădem orele (nu dă negativ pentru că T1>=T2)
*apoi ne ocupam separat de secunde daca a dat negativ,
*respectiv de minute daca a dat negativ
daca s1<0 atunci
      m ← m - 1 *luam un minut din minutele finale (chiar dacă m1<=0)
      s ← s + 60 *adaugam 1min=60sec.
daca m1<0 atunci
      h ← h - 1  *luam 1h
      m ← m + 60 *adaugam 1h=60min.
scrie h,m,s
```

Descrierea algoritmului în C++ (în CodeBlocks):

```
#include <iostream>
using namespace std;
int main()
{
 int h1,m1,s1,h2,m2,s2,h,m,s;
 cout<<"Dati T1 (ore-min-sec): "; cin>>h1>>m1>>s1;
 cout<<"Dati T2 (ore-min-sec): "; cin>>h2>>m2>>s2;
 s = s1-s2; //scadem secundele chiar dacă da negativ
 m = m1-m2; //scadem minutele chiar dacă da negativ
 h = h1-h2; //scadem orele (nu da negativ pentru ca T1>=T2)
 //apoi ne ocupam separat de secunde daca am obtinut val.neg.,
 //respectiv de minute daca s-a obtinut valoare negativa
 if (s<0)
 {
     m--; //luam 1 minut
     s += 60; //adaugam 1min = 60sec.
 }
 if (m<0)
 {
     h--; //luam 1h
     m += 60; //adaugam 1h=60sec.
 }
 cout<<"T1-T2 = "<<h<<"h "<<m<<"min. "<<s<<"sec."<<endl;
 return 0;
}
```

R3 (3* - suplimentar).

Enunţul problemei (exemplu de calculul valorii unei funcţii cu două ramificaţii): Pentru o valoare reală a lui x dată, să se calculeze valoarea funcţiei

$$f(x) = \begin{cases} e^{x-1}, & x < 2 \\ \log_3 x, & x \geq 2 \end{cases}.$$

Metoda de rezolvare: Valoarea lui x se compară cu 2 (dacă x este mai mic decât 2 sau mai mare sau egal) şi în funcţie de aceasta, valoarea funcţiei se calculează cu forma de pe una din cele două ramuri.
De exemplu, pentru $x = 1$: $f(x) = e^0 = 1$, iar pentru $x = 3$: $f(x) = \log_3 3 = 1$.

Descrierea algoritmului în pseudocod:

```
citeşte x
```

```
daca x<2  atunci
      y ← e^(x-1)
altfel
      y ← log3x
scrie y
```

sau fără utilizarea variabilei *y* pentru că nu mai este necesară mai departe

```
citeşte x
daca x<2 atunci
      scrie e^(x-1)
altfel *sigur x>=2
      scrie log3x
```

Observaţie: Următoarele funcţii matematice sunt descrise în biblioteaca standard <math.h> sau <cmath> a limbajelor de programare C / C++.

1. double **exp** (double *x*); → e^x
2. double **log** (double *x*); → logaritm natural: ln (*x*), pentru $x>0$
3. double **log10** (double *x*); → logaritm zecimal: lg (*x*), pentru $x>0$
4. double **pow** (double *x,* double *y*); → $x^y=e^{y \ln x}$, pt *x* si *y* reale. Eroare pentru $x=0$ şi $y \le 0$ sau pentru $x<0$ şi $y \notin \mathbf{Z}$.
5. double **pow10** (int *x*); → 10^x
6. double **sqrt** (double *x*); → rădăcina pătrată $\sqrt{x}$, pentru $x \ge 0$
7. double **ceil** (double *x*); → cel mai mic întreg $\ge x$, întors ca double (rotunjire prin adaos). Ex: ceil(3.2)=4, ceil (-3.2= -3).
8. double **floor** (double *x*); → cel mai mare întreg $\le x$, întors ca double (rotunjire prin lipsă). Ex: floor(3.2)=3, floor (-3.2= -4).
9. int **abs** (int *x*); → valoarea absolută pentru întregi Ex. abs(5)=5, abs(- 4) = 4
10. double **fabs** (double *x*); → valoarea absolută pentru numere reale Ex. fabs(5.2)=5.2, fabs(- 4.7) = 4.7
11. long **labs** (long *x*); → valoarea absolută pentru numere long

Toate funcţiile prezentate mai sus există şi pentru tipul **long double** în loc de double şi au numele anterior cu prefixul "**l**".

Câteva exemple de formule matematice scrise în C/C++:

Formula matematică	Corespondentul în C/C++
x^{25}	`pow(x,25)`
$\sqrt{x^2+3}$	`sqrt(x*x+3)`
$[x]$	`floor(x)`
$\{x\} = x-[x]$	`x - floor(x)`
e^{x+4}	`exp(x+4)`
$\ln(x+1)$	`log(x+1)`
$\lg(x)$	`log10(x)`
$\log_3(x)$, care este egal cu $\ln(x)/\ln(3)$	`log(x)/log(3)`

Pentru determinarea radicalului de ordinul 3 (similar pentru orice ordin impar) putem folosi funcţia C `pow(x,y)`, primul parametru fiind valoarea de sub radical, iar puterea fiind 1.0 / 3. Funcţia de sub radical poate fi şi negativă, dar funcţia `pow` dă eroare în cazul în care

primul parametru este real negativ şi al doilea nu este întreg, vom proceda astfel: se poate scoate forţat un minus în faţa radicalului. Deci:

$$\sqrt[3]{f(x)} \quad \rightarrow \quad \begin{cases} \text{pow}(f(x),1.0/3), & f(x) \geq 0 \\ -\text{pow}(-f(x),1.0/3), & f(x) < 0 \end{cases}$$

Descrierea algoritmului în C++ (în `CodeBlocks`) :

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
 float x,y; //y sau fx, nu f(x)
 cout<<"x=";  cin>>x;
 if (x<2) y = exp(x-1);
 else y = log(x)/log(3); //formula de schimbare de baza
 //log_a(x)=log_b(x)/log_b(a)
 cout<<"f(x)="<<y<<endl;
 return 0 ;
}
```

sau nefolosind variabila *y*:

```
#include<iostream>
#include<cmath>
using namespace std;
int main(){
 float x;
 cout<<"x=";  cin>>x;
 if (x<2) cout<<"f(x)="<<exp(x-1)<<endl;
 else cout<<"f(x)="<<log(x)/log(3)<<endl;
 return 0;
}
```

R4 (3* - suplimentar).

Enunţul problemei (exemplu de calculul valorii unei funcţii cu trei ramificaţii): Pentru o valoare reală a lui *x* dată, să se calculeze valoarea funcţiei:

$$f(x) = \begin{cases} [x], & x \geq 5 \\ \{x\}, & x \in (-1,5) \\ |x|, & x \leq -1 \end{cases}.$$

Metoda de rezolvare: Valoarea lui *x* se compară cu frontiera 5 şi apoi cu –1 şi în funcţie de acest lucru, valoarea funcţiei se calculează cu forma de pe una din cele trei ramuri. De exemplu: f(6.2)=[6.2]=6, f(7)=[7]=7, f(1.32)={1.32}=0.32, f(–0.3)={–0.3}= –0,3 – [–0.3]= = –0.3 – (–1) = 0.7, f(-2.3)=|-2.3|=2.3, f(-2)=|-2|=2.

Descrierea algoritmului în pseudocod:

```
citeşte x
daca x≥5 atunci
      scrie [x]
altfel *x<5=> (-∞,-1] sau (-1,5)
     daca x>-1 atunci
           scrie {x}
     altfel      *x<=-1
           scrie |x|
```

Descrierea algoritmului în C++(în CodeBlocks):

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
 float x;
 cout<<"x=";  cin>>x;
 if (x>=5) cout<<"f(x)="<<floor(x); //parte intreaga prin lipsa
 else //x<5 => x<=-1 sau -1<x<5
   if (x>-1) cout<<"f(x)="<<x-floor(x); //parte fractionara
   else cout<<"f(x)="<<fabs(x); //modul unui numar real
 return 0;
}
```

R5 (1*).

Enunţul problemei: Să se citească de la tastatură trei numere întregi şi să se stabilească dacă sunt pitagoreice sau nu.

Metoda de rezolvare: Trei numere naturale *a*, *b*, *c* sunt pitagoreice dacă verifică teorema lui Pitagora, adică:

$a^2 + b^2 = c^2$ sau $b^2 + c^2 = a^2$ sau $c^2 + a^2 = b^2$.

De exemplu:

a = 3, *b* = 4 şi *c* = 5 sunt pitagoreice,
a = 4, *b* = 3 şi *c* = 5 sunt pitagoreice
a = 1, *b* = 2 şi *c* = 3 nu sunt pitagoreice

Descrierea algoritmului în pseudocod:

```
citeşte a,b,c
daca a^2+b^2=c^2 sau b^2+c^2=a^2 sau c^2+a^2=b^2 atunci
      scrie "sunt numere pitagoreice"
altfel
      scrie "nu sunt numere pitagoreice"
```

Descrierea algoritmului în C++ (sub CodeBlocks):

```
#include <iostream>
using namespace std;
int main()
{
    int a,b,c;
    cout<<"a="; cin>>a;
    cout<<"b="; cin>>b;
    cout<<"c="; cin>>c;
    if ((a*a+b*b==c*c) || (b*b+c*c==a*a) || (c*c+a*a==b*b))
        cout<<"Numerele date sunt pitagoreice"<<endl;
    else
        cout<<"Numerele date nu sunt pitagoreice"<<endl;
    return 0;
}
```

R6 (3-4*).

Enunţul problemei: Se citesc valorile a 3 numere reale *a*, *b*, *c*. Să se descrie un algoritm prin care să se stabilească dacă pot forma laturile unui triunghi, iar în caz afirmativ să se stabilească ce tip de triunghi este: echilateral, dreptunghic-isoscel, isoscel, dreptunghic, oarecare.

Metoda de rezolvare: Ştim că într-un *triunghi*, suma oricăror două laturi este mai mare decât a treia latură. Va trebui să ţinem cont şi de faptul că lungimile laturilor sunt strict pozitive. Atunci, *a*, *b*, *c* pot forma laturile unui triunghi dacă:

$a+b>c$ şi $a+c>b$ şi $b+c>a$ şi $a>0$ şi $b>0$ şi $c>0$

Negarea proprietăţii de mai sus, adică a,b,c nu formează laturile unui triunghi dacă:

$a+b\le c$ sau $a+c\le b$ sau $b+c\le a$ sau $a\le 0$ sau $b\le 0$ sau $c\le 0$

Ştim că într-un triunghi *echilateral* toate laturile sunt egale, adică $a=b=c$. Dubla egalitate se va scrie ca două egalităţi care trebuie îndeplinite simultan:

$a = b$ şi $b = c$

Nu mai este necesară o a treia condiţie, $a=c$, căci aceasta rezultă din tranzitivitatea celor două egalităţi anterioare.

Într-un triunghi *isoscel*, două laturi sunt egale şi atât (nu sunt egale cu a treia):

($a = b$ şi $b \ne c$) sau ($b = c$ şi $c \ne a$) sau ($a = c$ şi $c \ne a$)

Într-un triunghi *dreptunghic*, *a*, *b*, *c* sunt numere pitagoreice:

$a^2 + b^2 = c^2$ sau $b^2 + c^2 = a^2$ sau $a^2 + c^2 = b^2$

Ar mai exista şi cazul când triunghiul este *dreptunghic-isoscel*, caz în care sunt îndeplinite cele două condiţii de mai sus.

În cazul în care nu este îndeplinită nicio condiţie din cele de mai sus, atunci triunghiul este isoscel.

De exemplu:

- pentru $a = 1$, $b = 2$ şi $c = 3$, acestea nu pot forma laturile unui triunghi, pentru că deşi 2+3>1 şi 1+3>2, nu este îndeplinită cea de-a treia: $1 + 2 \not> 3$;
- pentru $a = b = c = 2$ triunghiul este echilateral;
- pentru $a = 2$, $b = 2$ şi $c = 3$ triunghiul este isoscel;
- pentru $a = 6$, $b = 8$ şi $c = 10$ triunghiul este dreptunghic;
- pentru $a = 1$, $b = 1$ şi $c = \sqrt{2}$ triunghiul este dreptunghic-isoscel;
- pentru $a = 3$, $b = 4$ şi $c = 6$ triunghiul este oarecare.

Descrierea algoritmului în pseudocod:

```
citeşte a,b,c    *se citesc lungimile laturilor
daca a+b>c şi a+c>b şi b+c>a şi a>0 şi b>0 şi c>0 atunci
     scrie "Pot forma laturile unui triunghi"
     daca a=b şi b=c atunci
          scrie "Triunghiul este echilateral"
     altfel     *a, b, c nu sunt toate egale
     *testam intai proprietatea de "dreptunghic-isoscel" caci
     *este mai puternica (include) cazurile separate
          daca (a²+b²=c² sau b²+c²=a² sau a²+c²=b²) si
               (a=b sau b=c sau a=c) atunci
          *nu mai punem coditia per total "două laturi egale
          *si atat, pentru ca nu mai pot fi egale cu a treia
               scrie "Triunghiul este dreptunghic-isoscel"
          altfel *testam separat dreptunghic, resp. isoscel
               daca a=b sau b=c sau a=c atunci
                    scrie "Triunghiul este isoscel"
               altfel
                    daca a²+b²=c² sau b²+c²=a² sau a²+c²=b² atunci
                         scrie "Triunghiul este dreptunghic"
```

```
                    altfel
                        scrie "Triunghiul este oarecare"
altfel
    scrie "Nu pot forma laturile unui triunghi"
```

Descrierea algoritmului în C++ (CodeBlocks):

```
#include <iostream>
using namespace std;
int main()
{float a,b,c;
 int echilateral,isoscel,dreptunghic;
 //folosim variabile pentru a retine valoarile de adevar
 //ale conditiillor pentru tipurile triunghiurilor
 cout<<"Dati 3 numere reale: "; cin>>a>>b>>c;
 echilateral = (a==b && b==c);
 isoscel = (a==b || b==c || c==a);
 //forma simplificata caci va fi pe ramura "else"
 //a conditiei de triunghi echilateral
 dreptunghic =((a*a+b*b==c*c)||(a*a+c*c==b*b)||(b*b+c*c==a*a));
 if ( a+b>c && a+c>b && b+c>a && a>0 && b>0 && c>0)
 {  cout<<"Numerele pot forma laturile unui triunghi"<<endl;
    if (echilateral)  cout<<"Triunghi echilateral"<<endl;
    else
        if (dreptunghic && isoscel)
            cout<<"Triunghi dreptunghic-isoscel"<<endl;
        else
            if (isoscel) cout<<"Triunghi isoscel"<<endl;
            else
                if (dreptunghic)
                    cout<<"Triunghi dreptunghic"<<endl;
                else
                    cout<<"Triunghi oarecare"<<endl;
 }
 else
    cout<<"Numerele nu pot forma laturile unui triunghi"<<endl;
 return 0;
}
```

R7 (1* - suplimentar).

Enunţul problemei: Se introduc trei date de forma număr de ordine pacient, respectiv valoare glicemie. Descrieţi un algoritm pentru a afişa numărul de ordine al pacienţilor cu glicemia mai mare decât 100. Exemplu: pentru nr. 6 cu glicemie 90, nr. 10 cu glicemie 107 şi nr. 21 cu glicemie 110, se va afişa 10 21.

Metoda de rezolvare: Algoritmul constă în citirea a 3 perechi de date (la implementare ar fi indicat să se folosească structuri şi vectori, dar aici nu folosim) de genul număr de ordine – glicemie. Apoi, testăm fiecare glicemie în parte şi dacă depăşeşte valoarea 100 atunci se afişează numărul de ordine corespunzător.

Descrierea algoritmului în pseudocod:

```
citeşte n1,g1,n2,g2,n3,g3
*compararea glicemiilor cu 100 se face individual
*=> 3 if-uri separate
daca g1>100 atunci
    scrie n1
daca g2>100 atunci
    scrie n2
daca g3>100 atunci
    scrie n3
```

Descrierea algoritmului în C++ (sub CodeBlocks):

```
#include <iostream>
using namespace std;
int main(){
 int g1,n1,g2,n2,g3,n3;
 cout<<"Dati nr. de ordine si glicemia pt.1-ul pacient: ";
 cin>>n1>>g1;
 cout<<"Dati nr. de ordine si glicemia pt.al 2-lea pacient: ";
 cin>>n2>>g2;
 cout<<"Dati nr. de ordine si glicemia pt.al 3-lea pacient: ";
 cin>>n3>>g3;
 //urmeaza 3 if-uri separate:
 if (g1>100) cout<<n1<<" ";
 if (g2>100) cout<<n2<<" ";
 if (g3>100) cout<<n3<<" ";
 return 0;
}
```

Rulare:

```
Dati nr. de ordine si glicemia pt.1-ul pacient: 6 90 <Enter>
Dati nr. de ordine si glicemia pt.al 2-lea pacient: 10 107 <Enter>
Dati nr. de ordine si glicemia pt.al 2-lea pacient: 21 110 <Enter>
10 21
```

R8 (suplimentar).

Enunţul problemei: Să se descrie un algoritm pentru determinarea rădăcinile complexe ale ecuaţiei $ax^2 + bx + c = 0$, unde a, b şi c sunt valori reale cunoscute.

Metoda de rezolvare: Dacă $a \neq 0$, avem o ecuaţie de gradul al doilea şi se foloseşte metoda de rezolvarea a ecuaţiilor de gradul doi, folosind formulele pentru calculul discriminantului şi a rădăcinilor: $\Delta = b^2 - 4ac$ şi $x_{12} = \frac{-b \pm \sqrt{\Delta}}{2a}$. De exemplu, pentru a=1, b=3, c=2 ecuaţia are două rădăcini reale: $x_1 = -1$, $x_2 = -3$, pentru a=1, b=-6, c=9 ecuaţia are două rădăcini egale: $x_1 = x_2 = 3$, iar pentru a=1, b=2, c=3, Δ=-8, $\sqrt{-8} = i\sqrt{8} = 2i\sqrt{2}$ ecuaţia are două rădăcini complexe: $x_{12} = \frac{-2 \pm 2i\sqrt{2}}{2} = -1 \pm i\sqrt{2}$.

În cazul rădăcinilor complexe, acestea sunt conjugate: $x_{12} = \text{Re} \pm i \cdot \text{Im}$, unde $i = \sqrt{-1}$, partea reală este $\text{Im} = \frac{-b}{2a}$, iar partea imaginară este $\text{Im} = \frac{\sqrt{-\Delta}}{2a}$, cu $\Delta<0$.

Apoi, dacă a=0, atunci ecuaţia devine $bx+c = 0$. Rezultă că $x = -c / b$ pentru $b \neq 0$; în caz contrar, dacă $a = 0$ şi $b = 0$ ecuaţia devine c = 0 care este adevărată pentru orice x complex dacă $c = 0$, respectiv falsă dacă $c \neq 0$. De exemplu, pentru a=0, b=2, c=3 ecuaţia devine $2x + 3 = 0 \Rightarrow x = -1.5$, pentru a=0, b=0, c=2 ecuaţia devine 2=0 => ecuaţia nu are soluţii, iar pentru a=0, b=0, c=0 ecuaţia devine 0=0 => ecuaţia are soluţie orice x real.

Descrierea algoritmului în Pseudocod:

```
citeşte a,b,c          *se citesc datele de intrare
daca a ≠ 0 atunci      *ax²+bx+c=0 este ecuatie de grad 2
     Δ ← b²-4ac        *se calculeaza discriminantul
     *analiza semnului discriminantului
     daca Δ > 0 atunci     *ec.are 2 radacini reale diferite
          x1 ← (-b-√Δ)/2a
          x2 ← (-b+√Δ)/2a
          scrie x1,x2
     altfel
          daca Δ=0 atunci *ec.are 2 radacini reale egale
               x ← -b/2a
               scrie x
          altfel     *Δ<0 => ec.are 2 radacini complexe
               re ← -b/2a
               im ← √(-Δ)/2a
               scrie re ± i · im
altfel                 *a=0 => ec. devine: bx + c = 0
     daca b ≠ 0 atunci
          x ← -c/b
          scrie x
     altfel     *a=b=0 => ecuatia devine c=0
          daca c ≠ 0 atunci     *ecuatia: nenul=0
               scrie "Ecuatia nu are solutii"
          altfel                *a=b=c=0 => ecuatia devine 0=0
               scrie "Ecuatia are solutii orice x complex"
```

Descrierea algoritmului în C++ (în CodeBlocks):

```
#include <iostream>
#include <cmath> //sau math.h //pt sqrt
using namespace std;
int main()
{
 float a,b,c,delta,re,im;
 cout<<"Dati coeficientii ecuatiei: ";
 cin>>a>>b>>c;
 if (a) //sau (a!=0)
 {
    delta = b*b - 4*a*c;
    if (delta>0) //radacini complexe
    {
        cout<<"Ec.are 2 rad. reale diferite:"<<endl;
        cout<<"x1 = "<<(-b-sqrt(delta))/(2*a)<<endl;
        cout<<"x2 = "<<(-b+sqrt(delta))/(2*a)<<endl;
    }
```

```
   else
      if (delta == 0)
          cout<<"Ec.are 2 rad. egale: x1=x2="<<-b/(2*a)<<endl;
      else //delta<0
      {
          re = -b/(2*a); //sau -b/2/a;
          im = sqrt(-delta)/(2*a);
          cout<<"Ec.are 2 rad.complexe conjugate: "<<endl;
          cout<<"x1 = ("<<re<<") - i * ("<<im<<")"<<endl;
          cout<<"x2 = ("<<re<<") + i * ("<<im<<")"<<endl;
      }
   }
   else            //a=0 => ecuatia devine bx+c=0
      if (b)       //sau b!=0
          cout<<"Ecuatia are o solutie reala: "<<-c/b<<endl;
      else         //a=b=0 => ec. c=0
          if (c)   //sau (c!=0)
              cout<<"Ecuatia nu are solutii"<<endl;
          else
              cout<<"Ecuatia are solutii orice x complex"<<endl;
   return 0;
}
```

R9 (2*).

Enunţul problemei: Descrieţi un algoritm pentru a determinarea mediei generale de admitere a facultate după formula
medie_admitere = 75%*medie_bac +25%*max{nota_mate_bac, nota_info_bac}.

Metoda de rezolvare: Algoritmul constă în compararea notelor de bacalaureat de la matematică, respectiv informatică; în funcţie de care notă este mai mare aceea se va lua în considerare la calculul mediei de admitere. Reamintim că $\max\{a, b\} = \begin{cases} a, a \geq b \\ b, a < b \end{cases}$.

Descrierea algoritmului în pseudocod:

```
citeşte medie_bac,nota_mate_bac,nota_info_bac
daca nota_mate_bac ≥ nota_info_bac atunci
     medie_admitere ← 75%·medie_bac + 25%·nota_mate_bac
altfel  *este cazul: nota_mate_bac < nota_info_bac
     medie_admitere ← 75%·medie_bac + 25%·nota_info_bac
scrie medie_admitere
```

Descrierea algoritmului în C++ (sub CodeBlocks):

```
#include <iostream>
#include <iomanip> //pentru setprecision
using namespace std;
int main()
{
 float medie_bac,nota_mate_bac,nota_info_bac,medie_admitere;
 cout<<"Dati media de la bacalaureat: ";
 cin>>medie_bac;
 cout<<"Dati nota de la matematica de la bacalaureat: ";
 cin>>nota_mate_bac;
 cout<<"Dati nota de la informatica de la bacalaureat: ";
 cin>>nota_info_bac;
 if (nota_mate_bac>=nota_info_bac)
    medie_admitere = 0.75*medie_bac + 0.25*nota_mate_bac;
 else //nota_mate_bac<nota_info_bac
    medie_admitere = 0.75*medie_bac + 0.25*nota_info_bac;
```

```
       cout<<"Media de admitere = " << fixed << setprecision(2) <<
medie_admitere << endl;
       return 0;
      }
```

Rulare:

```
Dati media de la bacalaureat: 8.5 <Enter>
Dati nota de la matematica de la bacalaureat: 9 <Enter>
Dati nota de la informatica de la bacalaureat: 8 <Enter>
Media de admitere = 8.62
```

Observaţie: Dacă am fi utilizat

```
medie_admitere = 75/100*medie_bac + 25/100*nota_mate_bac
```

atunci rezultatul ar fi fost 0 căci s-ar fi calculat întâi 75/100 care face 0 în C++, apoi se calculează `25/100` care de asemenea face 0 şi astel rezultatul va fi 0 şi este memorat într-o variabilă de tip float (medie_admitere).

Am fi putut folosi, variante similare cu cele din programul de mai sus:

```
medie_admitere = 75.0/100*medie_bac + 25/100.0*nota_mate_bac;
```

(aici împărţirea este reală pentru că unul dintre operanzi este convertit la real)

```
medie_admitere = 75*medie_bac/100 + 25*nota_mate_bac/100;
```

(pentru că medie_bac este variabilă de tip real, atunci împărţirea este reală)

De asemenea, în loc de

```
       if (nota_mate_bac>=nota_info_bac)
          medie_admitere = 0.75*medie_bac + 0.25*nota_mate_bac;
       else //nota_mate_bac<nota_info_bac
          medie_admitere = 0.75*medie_bac + 0.25*nota_info_bac;
```

se poate folosi operatorul condiţional:

```
medie_admitere = (nota_mate_bac >= nota_info_bac) ?
0.75*medie_bac + 0.25*nota_mate_bac : 0.75*medie_bac +
0.25*nota_info_bac;
```

R10 (2-3* - suplimentar).

Enunţul problemei: Să se calculeze media aritmetică, geometrică şi armonică a două numere reale strict pozitive citite de la tastatură şi să se verifice inegalitatea mediilor.

Metoda de rezolvare: Pentru orice $a, b > 0$ avem: $m_a = \frac{a+b}{2}$, $m_g = \sqrt{ab}$, $m_{arm} = \frac{2}{\frac{1}{a}+\frac{1}{b}}$.

şi are loc inegalitatea mediilor este: $m_a \geq m_g \geq m_{arm}$ (dublă inegalitate = două inegalităţi).

De exemplu:

- pentru $a = 4$ şi $b = 9$: $m_a = 6.5$, $m_g = 6$, $m_{arm} = 5.54$ şi inegalitatea mediilor este adevarată.
- pentru $a = b = 3$: $m_a = 3$, $m_g = 3$, $m_{arm} = 3$ şi inegalitatea mediilor este adevarată.

Descrierea algoritmului în pseudocod:

```
citeşte a, b
daca a ≤ 0 sau b ≤ 0 atunci   *se verifica datele de intrare
     *daca unul dintre numere este necorespunzator
     scrie "date de intrare eronate"     *afisam un mesaj
     iesire                              *si iesim din program
ma ← (a+b)/2
mg ← √(ab)
```

```
marm ← 2/(1/a+1/b)
scrie ma,mg,marm
daca ma ≥ mg  si  mg ≥ marm  atunci     *dubla inegalitate
     scrie "este verificata inegalitatea mediilor"
altfel
     scrie "nu este verificata inegalitatea mediilor"
```

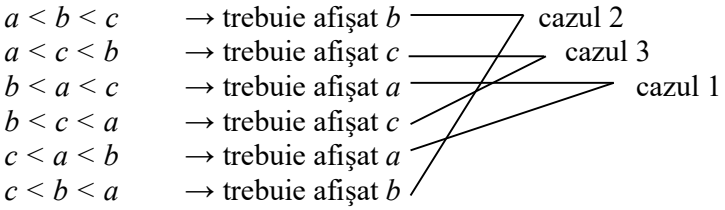
Descrierea algoritmului în C++ (sub CodeBlocks):

```
#include <iostream> //pt cin, cout
#include <iomanip>  //pt setprecision
#include <stdlib.h> //pt exit
#include <math.h>   //pt sqrt
using namespace std;
int main(){
 float a,b,ma,mg,marm;
 cout<<"a="; cin>>a;
 cout<<"b="; cin>>b;
 //testam daca a si b nu sunt strict pozitive
 if ((a<=0)||(b<=0)){ //sau (!(a>0 && b>0))
    cout<<"Date de intrare eronate";
    exit(0); //iesire din program
    //sau puteam folosi return 0; => iesire din functia main
   }
 ma = (a+b)/2;
 mg = sqrt(a*b);
 marm = 2/(1/a+1/b);
 cout<<"Media aritmetica: "<<setprecision(2)<<ma<<endl;
 cout<<"Media geometrica: "<<mg<<endl;
 cout<<"Media armonica: "<<marm<<endl;
 if ((ma>=mg)&&(mg>=marm))
   cout<<"Este verificata inegalitatea mediilor"<<endl;
 else
   cout<<"Nu este verificata inegalitatea mediilor<<endl";
 return 0;
}
```

R11. (suplimentar)

Enunţul problemei: Se dau trei numere diferite. Să se determine un algoritm pentru a determina numărul a cărei valoare este cuprinsă între celelalte două. De exemplu, pentru datele de intrare: 12 14 10 se va afişa 12.

Metoda de rezolvare: Notând cu *a*, *b* şi *c* cele trei numere diferite între ele, variantele posibile pentru valorile acestora sunt următoarele 6:



Din punctul de vedere al datelor de ieşire (se afişează *a*, *b* sau *c*) se pot reduce la 3 cazuri, fiecare caz având două condiţii cu operatorul sau între ele.

Dubla inegalitate se exprimă prin două inegalităţi; de exemplu $a<b<c \Leftrightarrow a<b$ şi $b<c$.

Descrierea algoritmului în pseudocod:

```
citeste a,b,c    *a≠b≠c≠a
daca (b<a si a<c) sau (c<a si a<b) atunci   *cazul 1
     scrie a
```

```
altfel
     daca (a<b si b<c) sau (c<b si b<a) atunci   *cazul 2
          scrie b
     altfel      *cazul 3 - cel rămas
          scrie c
```

Descrierea algoritmului în C++(CodeBlocks):

```
#include <iostream>
using namespace std;
int main()
{
 int a,b,c;
 cout<<"a = "; cin>>a;
 cout<<"b = "; cin>>b;
 cout<<"c = "; cin>>c;
 if ((b<a)&&(a<c))||((c<a)&&(a<b)) //cazul 1 cand se afis.a
   cout<<a;
 else
   if (((a<b)&&(b<c))||((c<b)&&(b<a))) //cazul 2 cand se afis.b
     cout<<b;
   else //cazul 3 cand se afiseaza c
    cout<<c;
 return 0;
}
```

Rulare:

```
a = 12 <Enter>
b = 14 <Enter>
c = 10 <Enter>
12
```

Câteva explicaţii cod C++:

Spunem că vom considera ca făcând parte din cazul 1: $b < a < c$ (care înseamnă $b<a$ şi $a<c$) sau $c < a < b$ (care înseamnă $c<a$ şi $a<b$). În C/C++ acestea s-au scris astfel:

```
if ( ((b<a)&&(a<c)) || (((c<a)&&(a<b)) ) ...
```

R12. (suplimentar)

Enunţul problemei: Se consideră 3 valori reale reţinute în variabilele *a*, *b* şi *c*. (*i*) Să se determine cea mai mică valoare dintre cele trei valori. (*ii*) Să se afişeze cele trei valori în ordine crescătoare.

Descrierea algoritmului:

(*i*) O prima variantă de rezolvare poate fi cea bazată pe compararea valorilor două câte două:

```
citeşte a,b,c    *se citesc datele de intrare
daca a≤b atunci
     daca a≤c atunci min ← a    *a≤b si a≤c
     altfel min ← c             *c<a≤b
altfel
     daca b≤c atunci min ← b    *b<a si b≤c
     altfel min ← c             *c<b<a
scrie min        *afisarea rezultatului
```

O altă variantă, mai compactă, şi care poate fi uşor extinsă la cazul mai multor valori este cea în care se presupune ca valoarea minimă este chiar prima, după care se compară cu următoarele valori, iar în momentul identificării unei valori mai mici valoarea minimă este actualizată.

$\min\{a,b,c\} = \min\{\min\{a,b\},c\}$

```
citeşte a,b,c    *se citesc datele de intrare
min ← a          *valorea initiala a minimului se pp. a fi a
daca b<min atunci min ← b
daca c<min atunci min ← c
scrie min        *afisarea rezultatului final
```

(*ii*) O primă variantă poate fi aceea în care se identifică toate cele şase variante asociind câte o condiţie fiecăreia:

```
citeşte a,b,c    *se citesc datele de intrare
daca a<=b si b<=c atunci scrie a,b,c
daca a<=c si c<b  atunci scrie a,c,b
daca c<a  si a<=b atunci scrie c,a,b
daca c<=b si b<a  atunci scrie c,b,a
daca b<a  si a<=c atunci scrie b,a,c
daca b<c  si c<a  atunci scrie b,c,a
```

O a doua variantă este aceea în care se compară succesive câte două valori:

```
citeşte a,b,c
daca a≤b atunci
     daca b≤c atunci scrie a,b,c
     altfel     *c<b
          daca a≤c atunci scrie a,c,b
          altfel scrie c,a,b
altfel  *b<a
     daca a≤c atunci scrie b,a,c
     altfel     *c<a
          daca b≤c atunci scrie b,c,a
          altfel scrie c,b,a
```

R13 (1-2*).

Enunţul problemei: Să se scrie un algoritm pentru calculul notei finale la Examenul de Algoritmi şi structuri de date.

Metoda de rezolvare:

Să ne reamintim că pentru calculul notei finale vor conta:

- nota la partea de Algoritmi (s-o notăm cu $nota_1$)
- nota la partea de Structuri de Date (s-o notăm cu $nota_2$)
- nota la temele pentru acasă (s-o notăm cu $nota_{teme}$)

Deci datele de intrare sunt: $nota_1$, $nota_2$ şi $nota_{teme}$.

Valoarea notei finale se determină astfel:

$$notafinala = \begin{cases} 40\% * nota_1 + 40\% * nota_2 + 20\% * nota_{teme}, & daca\ nota_1, nota_2 \geq 5 \\ 4, & altfel \end{cases}$$

Descrierea algoritmului în pseudocod:

```
citeste nota1, nota2, notateme
daca nota1 ≥ 5 şi nota2 ≥ 5 atunci
     notafinala ← 40%*nota1 + 40%*nota2 + 20%*notafinala
altfel
     notafinala ← 4
scrie notafinala
```

Se poate evita folosirea variabilei $nota_{finală}$, afisându-se direct rezultatul expresiei notei finale:

```
citeste nota1, nota2, notateme
daca nota1 ≥ 5 şi nota2 ≥ 5 atunci
     scrie 40%*nota1 + 40%*nota2 + 20%*notateme
*altfel
     scrie 4
```

Descrierea algoritmului în C++:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
 int nota_1,nota_2,nota_teme;
 cout<<"Nota_1 = "; cin>>nota_1;
 cout<<"Nota_2 = "; cin>>nota_2;
 cout<<"Nota_teme = "; cin>>nota_teme;
 if (nota_1>=5 && nota_2>=5)
cout<<fixed<<setprecision(2)<<0.4*nota_1+0.4*nota_2+0.2*nota_teme;
 else cout<<4;
 return 0;
}
```

După rularea programului, pe ecran va apare:

```
Nota_1 = 9 <Enter>
Nota_2 = 8 <Enter>
Nota_teme = 10 <Enter>
8.80
```

respectiv:

```
Nota_1 = 6 <Enter>
Nota_2 = 4 <Enter>
Nota_teme = 10 <Enter>
4
```

Câteva explicaţii cod C++: Notele le-am declarat de tip întreg "int", iar la calculul notei finale, în loc de 40%*nota_1 +…. am scris 0.4*nota_1 +….

R14 (1*).

Enunţul problemei: Să se determine un algoritm pentru a verifica dacă un număr natural dat este divizibil sau nu, folosind criteriul de divizibilitate cu 25 (*n* se divide cu 25 ⇔ ultimele două cifre ale lui *n* formează un număr divizibil cu 25).

Metoda de rezolvare: Algoritmul va avea ca date de intrare un număr natural, sa-l notăm cu *n*. Numărul format din ultimele 2 cifre = restul împărţirii lui *n* la 100. Vom verifica dacă acest număr este sau nu divizibil cu 25 (adică dacă se împarte exact la 25 – altă variantă să verificăm dacă este egal cu 00 sau 25 sau 50 sau 75) şi în funcţie de aceasta vom scrie un mesaj de genul "numărul dat este / nu este divizibil cu 25).

Descrierea algoritmului în pseudocod:

```
citeste n  *n natural
daca (n%100)%25=0 atunci    *ultimele 2 cifre se divid cu 25?
      scrie "n se divide cu 25"
altfel
      scrie "n nu se divide cu 25"
```

Descrierea algoritmului în C++:

```
#include <iostream>
using namespace std;
int main()
{
 unsigned long long n; //valoare intreaga cat mai mare
 cout<<"n"; cin>>n;
 if (n%100%25 == 0) //sau (!(n%100%25))
   //ultimele 2 cifre se divid cu 25
   cout<<"n se divide cu 25"<<endl;
 else cout<<"n nu se divide cu 25"<<endl;
 return 0;
}
```

După rularea programului C++, pe ecran va apare:

```
n = 416275 <Enter>
n se divide cu 25
```

sau

```
n = 2300627 <Enter>
n nu se divide cu 25
```

Explicaţii cod C++:

Codul începe cu includerea fişierelor header necesare pentru a utiliza anumite funcţii în programul curent; de exemplu, aici am inclus `iostream` pentru că acolo este declarată funcţia de ieşire `cout` cu ajutorul căreia se pot afişa mesaje şi valorile unor variabile pe ecran.

Orice program C/C++ trebuie să aibă o funcţie principală `main`, aici aceasta nu are parametri (între paranteze nu este trecut nimic – similar se poate scrie `void main(void)`) şi nu returnează nimic (pentru că înainte apare cuvântul cheie `void`).

Linia `unsigned long long n;` reprezintă o declaraţie a variabilei *n*, local în funcţia main. Pentru că se testează restul împărţirii lui *n* la 100, trebuie declarată de un tip întreg – în C/C++ sub CodeBlocks acestea sunt cele din *ASD02Tipuri de date*". Am preferat tipul `unsigned long long` pentru că are domeniul cu marginea naturală superioară cea mai mare (pe mulţimea nr. naturale) şi va permite testarea corectă a oricărui număr *n* <= 18.446.744.073.709.551.615.

Se citeste valoarea lui *n* (`cin>>n`), anterior afişând pe ecran mesajul "*n*=" (cu `cout<<"n=";` pentru a se şti (cât de cât) ce trebuie introdus.

Apoi pentru testarea divizibilităţii ultimelor două cifre cu 25 am folosit condiţia să se împartă exact la 25: `n%100%25 == 0`. Aici avem trei operatori, doi operatori modulo (rest împărţire) şi operatorul de egalitate. Operatorii modulo au prioritate şi aceştia se execută în C++ de la stânga la dreapta (vezi: *ASD03Ordinea operatorilor*). Aşadar, întâi se determină n%100 (adică numărul format din ultimele două cifre ale lui *n* – ca tip de date acesta trebuia să fie întreg) apoi se determină … %25 (care este 0 dacă numărul format din ultimele două cifre este divizibil cu 25 şi un număr nenul în caz contrar) şi la final se compară dacă rezultatul este egal cu 0. De aceea, nu sunt necesare paranteze ca în `(n%100)%25 == 0`.

Ca variabile am folosit una singură; se mai putea reţine numărul format din ultimele două cifre într-o altă variabilă, să zicem *m* şi testarea acestuia din urmă dacă este divizibil cu 25:

```
int m = n%100;
if (m%25==0) cout << "n este divizibil cu 25"<<endl;
else cout << "n este divizibil cu 25"<<endl;
```

R15 (1*).

Enunţul problemei: Se dau vârstele a doi copii, exprimate în ani. Descrieţi un algoritm pentru a afişa care copil este mai mare şi cât este diferenţa de vârstă dintre cei doi copii. În caz de egalitate se va scrie "varste egale".
De exemplu: pentru 6 ani şi 13 ani se va afişa "al doilea copil este mai mare", cu 7 ani, pentru 7 ani şi 5 ani se va afişa "primul copil este mai mare", cu 2 ani, iar pentru 4 ani şi 4 ani se va afişa "varste egale".

Metoda de rezolvare: Notând cu v_1 şi v_2 vârstele celor doi copii, va trebui să testăm cele trei cazuri posibile: $v_1<v_2$, $v_1>v_2$ şi $v_1=v_2$.

Descrierea algoritmului în pseudocod:

```
citeşte v1,v2
daca v1<v2 atunci
      scrie " Al doilea copil este mai mare"
      scrie v2-v1            *se da diferenta in valoare absoluta
altfel  *acum v1≥v2, adica v1>v2 sau v1=v2
     daca v1>v2 atunci
           scrie "Primul copil este mai mare"
           scrie v1-v2
     altfel      *este clar ca v1=v2
           scrie "Varste egale"
```

Descrierea algoritmului în C++ (CodeBlocks):

```
#include <iostream>
using namespace std;
int main(){
 int v1,v2;
 cout<<"Dati varsta primului copil (in ani): "; cin>>v1;
 cout<<"Dati varsta celui de-al doilea copil (in ani): ";
 cin>>v2;
 if (v1<v2)
    cout<<"Al doilea copil este mai mare cu "<<v2-v1<<" ani.";
 else //acum v1>=v2 => v1>v2 sau v1=v2
    if (v1>v2)
        cout<<"Primul copil este mai mare cu "<<v1-v2<<" ani.";
    else //este clar: v1=v2
        cout<<"Varste egale.";
 cout<<endl;
 return 0; }
```

Rulare:

```
Dati varsta primului copil (in ani): 6
Dati varsta celui de-al doilea copil (in ani): 13
Al doilea copil este mai mare cu 7 ani.
sau
Dati varsta primului copil (in ani): 7
Dati varsta celui de-al doilea copil (in ani): 5
Primul copil este mai mare cu 2 ani.
sau
Dati varsta primului copil (in ani): 4
Dati varsta celui de-al doilea copil (in ani): 4
Varste egale.
```

R16 (2*).

Enunţul problemei: Să se descriere un algoritm ce simulează un calculator de buzunar: se introduc două numere întregi nenule şi o operaţie de tipul +, -, *, /, % reprezentând adunarea, scăderea, înmulţirea, câtul, respectiv restul împărţirii primului număr la al doilea; să se afişeze rezultatul.

Descrierea algoritmului în pseudocod:

```
citeste a, b, c  *presupunem ca a si b sunt nenule, c operator
alege c dintre
      '+':  scrie a+b
      '-':  scrie a-b
      '*':  scrie a*b
      '/':  scrie [a/b]     *catul impartirii lui a la b
      '%':  scrie a-[a/b]*b *restul impartirii lui a la b
rest: scrie "operator gresit"
```

Descrierea algoritmului în C++(folosind CodeBlocks):

```
#include <iostream>
using namespace std;
int main()
{
 int a,b;
 char c;
 cout<<"Dati doua numere naturale: ";
 cin>>a>>b;
 cout<<"Dati un operator (+,-,*,/,%): ";
 cin>>c;
 switch (c)
 {
  case '+': cout<<a<<c<<b<<" = "<<a+b; break;
  case '-': cout<<a<<c<<b<<" = "<<a-b; break;
  case '*': cout<<a<<c<<b<<" = "<<a*b; break;
  case '/': cout<<a<<c<<b<<" = "<<a/b; break; //catul
  case '%': cout<<a<<c<<b<<" = "<<a%b; break; //restul
  default: cout<<"Operator gresit";
 }
 cout<<endl;
 return 0;
}
```

R17 (2*).

Enunţul problemei: Să se descriere un algoritm ce stabileşte în ce anotimp se încadrează o lună calendaristică dată prin numarul său de ordine. De exemplu, luna numărul 5 este lună de primăvară, iar luna numărul 12 este lună de iarnă.

Descrierea algoritmului în pseudocod:

```
citeste nr_luna *numarul de ordine al unei luni
alege nr_luna dintre
     3,4,5:  scrie "primavara"  *dupa afisaj se iese din alege
     6,7,8:  scrie "vara"       *dupa afisaj se iese din alege
     9,10,11:  scrie "toamna"   *dupa afisaj se iese din alege
     12,1,2:  scrie "iarna"     *dupa afisaj se iese din alege
rest: scrie "numar de luna gresit "
```

Descrierea algoritmului în C++(folosind CodeBlocks):

```
#include <iostream>
using namespace std;
int main() {
 int nr_luna;
 cout<<"Dati numarul lunii: "; cin>>nr_luna;
 switch (nr_luna)
 {case 3:
  case 4:
  case 5: cout<<"primavara"; break;
  case 6:
  case 7:
  case 8: cout<<"vara"; break;
  case 9:
  case 10:
  case 11: cout<<"toamna"; break;
  case 12:
  case 1:
```

```
        case 2: cout<<"iarna"; break;
        default: cout<<"Nr luna gresita";
       }
       return 0;
      }
```

R18 (4*).

Enunţul problemei: Se citeşte de la tastatură o dată alcătuită din zi, luna, an. Să se afişeze numărul de zile trecute de la începutul acelui an.

Metoda de rezolvare: Întâi se se consideră zilele din luna curentă, apoi se însumează zilele din lunile anterioare. Va trebui să se ţină cont şi de faptul că anul curent este bisect sau nu în cazul în care va trebui să luăm în calcul şi luna februarie.
De exemplu, pentru 1 feb 2009 a trecut doar luna ianuarie întreagă şi deci sunt 31 de zile trecute; pentru 5 martie 2008 au trecut 31 zile în luna ianuarie, 29 de zile în luna februarie (2008 este an bisect) şi mai avem 4 zile trecute în luna curentă, martie => 31 + 29 + 4 = 64.

Descrierea algoritmului în C++(folosind CodeBlocks):

```
#include <iostream.>
using namespace std;
int main() {
 int zi,luna,an,zile;
 cout<<"Dati o data valida (zi luna an): ";
 cin>>zi>>luna>>an;
 //datele se introduc despartite de cate un spatiu
 zile = zi-1; //insumam zilele din luna curenta
 switch (luna-1) //insumam zilele din lunile anterioare
 {
  case 11: zile += 30; //nov
  case 10: zile += 31; //oct
  case  9: zile += 30; //sept
  case  8: zile += 31; //august
  case  7: zile += 31; //iul
  case  6: zile += 30; //iun
  case  5: zile += 31; //mai
  case  4: zile += 30; //apr
  case  3: zile += 31; //martie
  case  2: zile += 28; //feb pt an normal, nu bisect
  case  1: zile += 31; //ian
 }

 //an bisect si contorizam si luna feb =>
 //feb are de fapt 29 zile
 if ( luna>2 && ((an%4==0&&an%100!=0)||(an%400==0)) )
   zile++;
 cout<<"Nr de zile trecute de la inceputul anului:
"<<zile<<endl;
 return 0;
}
```

Rulare:

Dati o data valida (zi luna an): 1 2 2009 <Enter>
Nr de zile trecute de la inceputul anului: 31

sau

Dati o data valida (zi luna an): 5 3 2008 <Enter>
Nr de zile trecute de la inceputul anului: 64

TEMĂ PENTRU ACASĂ

Pentru începători:

1. Să se descrie un algoritm în C/C++ pentru calculul ariei unui trapez când se cunosc baza mică (b), baza mare (B) şi înălţimea (h). De exemplu, pentru b=3, B=6 şi h=5 se obţine A = (B+b)*h/2 = 9*5/2 = 22,5.
2. Se citesc valorile lungimii şi lăţimii unui dreptunghi. Să se descrie un algoritm în C/C++ pentru a calcula perimetrul şi aria dreptunghiului. De exemplu, pentru lungime=5 şi latime=3, se obţine P=2*(lungime+latime)=16, iar A=lungime*latime=15.
3. Ana vrea să verifice dacă greutatea şi înălţimea ei corespund vârstei pe care o are. Ea a găsit într-o carte următoarele formule de calcul ale greutăţii şi înălţimii unui copil, *v* fiind vârsta: greutate = 2**v* + 8 (în kg), înălţime = 5**v* + 80 (în cm). Descrieţi un algoritm (în pseudocod şi/sau C/C++) care să citească vârsta unui copil şi să afişeze greutatea şi înălţimea "ideală", folosind aceste formule. De exemplu, pentru *v* = 10 (ani) se obţine *g* = 28 (kg) şi *h* = 130 (cm).
4. Să se descrie un algoritm (în C/C++) calculul ariei şi lungimii unui cerc când se cunoaşte raza acestuia (aria = π*raza^2, lungime = 2*π*raza). De exemplu, pentru raza = 1 se obţine aria = 3.14, iar lungime = 6.28.
5. Să se descrie un algoritm (în C/C++) pentru a calcula aria şi perimetrul unui triunghi dreptunghic când se cunosc lungimile catetelor ($A = c_1c_2 / 2$, $ip = \sqrt{(c_1)^2 + (c_2)^2}$, $P = c_1 + c_2 + ip$). De exemplu, pentru c_1 = 6cm, c_2 = 8cm se obţine A = 24 cm^2, iar P = 24 cm.
6. Să se descrie un algoritm (în C/C++) pentru a determina înălţimea şi aria unui triunghi echilateral de latură cunoscută (h = lat $\cdot\sqrt{3}$ / 2, A = lat^2 $\cdot\sqrt{3}$ / 4). De exemplu, pentru lat = 2 cm se obţine $h = \sqrt{3}$ = 1,7 cm şi $A = \sqrt{3}$ = 1,7 cm^2.
7. Descrieţi un algoritm (în pseudocod şi/sau C/C++) pentru conversia unui unghi din grade în radiani. De exemplu, pentru x = 0 grade, valoarea în radiani este tot 0, iar pentru x = 180 grade, valoarea în radiani este 3.14. Sugestie: folosiţi formula de mai jos
$$\frac{x\_grade}{180} = \frac{x\_radiani}{\pi} \Rightarrow x\_radiani = x\_grade \cdot \pi / 180$$
8. Descrieţi un algoritm care să stabilească dacă un unghi dat (în grade) este unghi obtuz (>90^0), ascuţit (<90^0) sau drept. De exemplu, un unghi de 30 de grade este ascuţit, un unghi de 120 de grade este obtuz, iar un unghi de 90 de grade este drept.
9. Descrieţi un algoritm în care ştiind valoarea unghiurilor unui triunghi să se verifice proprietatea că "suma unghiurilor într-un triunghi este 180^0", apoi să determine tipul triunghiului: ascuţitunghic (toate unghiurile < 90^0), obtuzunghic (un unghi > 90^0) sau dreptunghic (un unghi = 90^0). De exemplu, pentru $\hat{A}$=60^0, $\hat{B}$=70^0 şi $\hat{C}$=50^0 triunghiul este ascuţitunghic, pentru $\hat{A}$=20^0, $\hat{B}$=120^0 şi $\hat{C}$=40^0 triunghiul este obtuzunghic, iar pentru $\hat{A}$=60^0, $\hat{B}$=90^0 şi $\hat{C}$=30^0 triunghiul este dreptunghic.
10. Un lift pentru copii acceptă o greutate de maxim 100kg. Citind de la tastatură greutăţile a doi copii, să se descrie un algoritm prin care să afişaţi dacă "pot intra ambii copii" sau "pot intra pe rând". Exemple: date de intrare: 87, 50, date de ieşire: intră pe rând, date de intrare 45 52, date de ieşire "pot intra ambii copii.
11. Se dau trei numere x, y, z. Să se descrie un algoritm pentru a scădea z din cel mai mare dintre *x* şi *y*.
12. Se dau numerele întregi a, b şi k. Să se descrie un algoritm pentru a stabili dacă fracţia *a* / *b* poate fi simplificată prin *k*. De exemplu, 6/4 poate fi simplificată prin 2, dar 4/9 nu poate fi simplificată prin 2.

13. Ionel are voie să se uite la TV 20 de ore pe săptămână. Se introduc numărul de ore când el se uită la TV pe fiecare zi din săptămână. Să se verifice dacă va fi pedepsit sau nu. Date de intrare: 3 4 2 2 5 6 1, date de ieşire: va fi pedepsit.
14. Să se descrie un algoritm pentru a verifica criteriul de divizibilitate cu 8: un număr se divide cu 8 dacă ultimele 3 cifre ale sale se divid cu 8. De exemplu, 23144 se divide cu 8, iar 35102 nu se divide cu 8.
15. Ionel spune părinţilor doar notele mai mari sau egale cu 7. Într-o zi, el a luat trei note. Citiţi-le de la tastatură şi afişaţi acele note pe care le va comunica şi părinţilor. De exemplu, date de intrare 8, 7, 5; date de ieşire: 8, 7.

Avansați:

1) Se citesc trei numere întregi. Să se descrie un algoritm pentru a verifica dacă sunt în progresie aritmetică (a, b, c în progresie aritmetică, dacă $b = (a+c)/2$). De exemplu, a=3, b=5 şi c=7 sunt în progresie aritmetică, a=9, b=6 şi c=3 sunt în progresie aritmetică, iar a=1, b=3 şi c=6 nu sunt în progresie aritmetică.
2) Se citesc 2 numere. Să se descrie un algoritm pentru a verifica dacă ele sunt consecutive. De exemplu 3 şi 4 sunt consecutive, dar 7 şi 6 nu sunt consecutive şi 3 şi 5 nu sunt consecutive.
3) Se citesc 3 numere. Să se descrie un algoritm pentru a verifica dacă acestea sunt consecutive? De exemplu, 2, 3 şi 4 sunt consecutive, 2, 4, 3 nu sunt consecutive şi 4, 3, 2 nu sunt consecutive şi 2, 3, 5 nu sunt consecutive.
4) Se introduc două numere. Să se descrie un algoritm prin care, care dacă al doilea număr este diferit de 0, să afişeze valoarea raportului dintre primul şi al doilea, iar dacă este nul să afişeze mesajul "împărţire imposibilă". De exemplu: Date de intrare: 10 3 => date de ieşire: 3.33. Date de intrare: 45 0 => date de ieşire “împărţire imposibilă”.
5) La un concurs se dau ca premii primilor 100 de concurenţi, tricouri de culoare albă, roşie, albastră şi neagră în această secvenţă (variantă: câte 25 de fiecare culoare). Ionel este pe locul x. Să se descrie un algoritm prin care să se stabilească ce culoare va avea tricoul pe care-l va primi? De exemplu, pentru x=38, se va afişa “tricou roşu”.
6) Cunoscând data curentă exprimată prin trei numere întregi reprezentând anul, luna, ziua şi precum şi data naşterii unei persoane, exprimată astfel, să se facă descrie un algoritm prin care să calculeze vârsta persoanei respective în număr de ani împliniţi. De exemplu, pentru data 2014 10 25 şi data naşterii 1969 11 2, se va afişa 44 de ani.
7) Să se descrie un algoritm prin care să se stabilească dacă un număr natural este pătrat perfect sau nu.
8) La ferma de găini este democraţie. Fiecare găină primeşte exact acelaşi număr de boabe de porumb. Cele care nu pot fi împărţite vor fi primite de curcan. Să se descrie un algoritm pentru a stabili cine a primit mai multe boabe şi cu cât; în caz de egalitate, se va afisa numărul de boabe primite şi cuvântul "egalitate". Datele se vor citi în ordinea: numărul de găini, iar după aceea numărul de boabe de porumb. De exemplu, pt 100 şi 4050 => curcanul are mai mult cu 10 boabe.
9) Într-o tabără, băieţii sunt cazaţi câte 4 într-o căsuţă, în ordinea sosirii. Ionel a sosit al n-lea. Să se descrie un algoritm pentru a stabili în a câta căsuţă se va afla Ionel? De exemplu, pentru n = 61 se va afişa căsuţa 16.
10) "Mă iubeşte un pic, mult, cu pasiune, la nebunie, deloc, un pic, ...". Rupând petalele unei margarete cu x petale, să se descrie un algoritm pentru a verifica ce urmează la “el(ea) mă iubeşte”. De exemplu, pentru x = 10 date de ieşire: el(ea) mă iubeşte de loc.
11) Să se descrie un algoritm pentru a calcula reuniunea a două intervale deschise. De exemplu, pentru intervalele (1, 2) şi (3, 4) se va afişa (1, 2) U (3, 4), dar pentru (1, 4) şi (2, 5) se va afişa (1, 5), iar pentru (1, 4) şi (2, 3) se va afişa (1, 4).