Curs 2: Descrierea algoritmilor
- Limbaj algoritmic
- Specificarea datelor
- Exemple
- Tehnica rafinării succesive şi subalgoritmi

Metodele de rezolvare a problemelor sunt adesea descrise în limbaj matematic. Deşi riguros, acesta nu este întotdeauna adecvat întrucât nu permite specificarea unor detalii importante in etapa codificării într-un limbaj de programare. Pentru descrierea algoritmilor se folosesc:

- *Scheme logice.* Permit descrierea algoritmilor în manieră grafică. Fiecărui tip de prelucrare simplă îi corespunde un simbol grafic iar prelucrările structurate sunt constituite din mai multe simboluri grafice conectate. Ordinea de efectuare a prelucrărilor este marcată prin săgeţi.
- *Limbaj algoritmic.* Este un limbaj artificial constituit dintr-un vocabular restrâns ce conţine cuvinte cheie asociate prelucrărilor şi identificatori ce permit denumirea datelor. Ca orice limbaj se bazează pe un alfabet, un vocabular şi un set de reguli de sintaxă. Regulile de sintaxă sunt mai puţin stricte decât în cazul limbajelor de programare. Sunt acceptate expresii descrise în manieră matematică. Limbajul de descriere a algoritmilor este denumit uneori *pseudocod* pentru a sugera atât apropierea de limbajele de programare însă şi absentă unor reguli de sintaxă foarte stricte.

1 Limbaj algoritmic

Pseudocodul pe care îl vom folosi în continuare permite specificarea prelucrărilor simple şi structurate după cum urmează.
Atribuire. Pentru atribuirea valorii obţinute prin evaluarea unei expresii variabilei cu numele v se specifică:

$$v \leftarrow < \text{expresie} >$$

Expresiile se utilizează, de regulă, pentru a descrie calcule şi sunt constituite din *operanzi* şi *operatori*. Operanzii pot fi variabile şi valori constante. Operatorii utilizaţi sunt:

- *aritmetici:* $+$ (adunare), $-$ (scădere), $*$ (înmulţire), / (împărţire), $\hat{}$ (ridicare la putere), DIV (câtul împărţirii întregi), MOD (restul împărţirii întregi);
- *relaţionali:* $=$ (egal), $\neq$ sau $<>$ (diferit), $<$ (strict mai mic), $\leq$ sau $<=$ (mai mic sau egal), $>$ (strict mai mare), $\geq$ sau $>=$ (mai mare sau egal);
- *logici:* OR (disjuncţie), AND (conjuncţie), NOT (negaţie).

Citire. Pentru completarea variabilei cu numele v cu o valoare preluată de la dispozitivul de intrare se specifică:

`READ` v

Scriere. Pentru transmiterea către dispozitivul de ieşire a valorii unei variabile sau a celei obţinute prin evaluarea unei expresii se specifică:

`WRITE` < expresie >

În specificarea oricărui algoritm este suficientă utilizarea următoarelor structuri: *secvenţială*, *condiţională* şi *repetitivă*.

Structura secvenţială. O succesiune de n prelucrări se specifică:

< prelucrare 1 >

< prelucrare 2 >

$\vdots$

< prelucrare n >

Structuri condiţionale. Se utilizează pentru specificarea prelucrărilor în a căror execuţie se urmează o ramură sau alta în funcţie de satisfacerea sau nesatisfacerea unei condiţii. O structură condiţională cu două variante de prelucrare se descrie prin:

`IF` < conditie > `THEN` < prelucrare 1 > `ELSE` < prelucrare 2 >

La execuţia acestei structuri, dacă prin evaluarea condiţiei se obţine valoarea *adevărat* atunci se execută prima prelucrare, altfel se efectuează cea de a doua. Cazul particular al unei singure variante se descrie prin:

`IF` < conditie > `THEN` < conditie >

În acest caz prelucrarea specificată se efectuează doar dacă condiţia este adevărată, altfel se trece direct la prelucrarea următoare din algoritm.

Structurile alternative sunt ilustrate în figura 1.

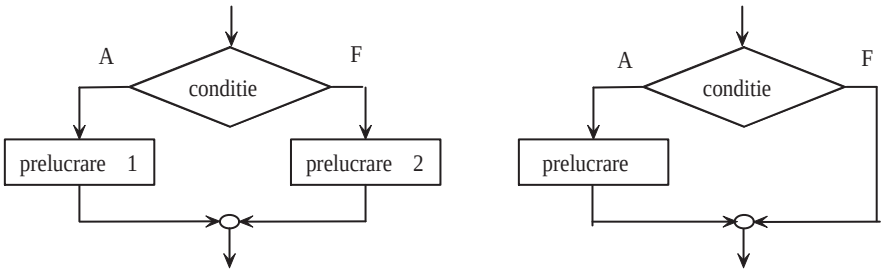

Figura 1: Descrierea grafică a structurilor alternative

Structuri repetitive. Permit descrierea prelucrărilor ce trebuie efectuate în mod repetat. În funcţie de modul de plasare a condiţiei de continuare (sau de oprire) există două variante de structuri repetitive: condiţionată anterior şi condiţionată posterior.

Varianta condiţionată anterior se specifică prin:

`WHILE` < conditie > `DO` < prelucrare >

La execuţie, prelucrarea se repetă atât timp cât condiţia este adevărată. Dacă condiţia este de la început falsă prelucrarea nu se efectuează nici o dată. Dacă in cadrul prelucrării nu se modifică componente ale condiţiei astfel încât aceasta să devină falsă atunci prelucrarea va fi repetată la nesfârşit.

Descrierea grafică a structurii repetitive condiţionată anterior este ilustrată în figura 2.

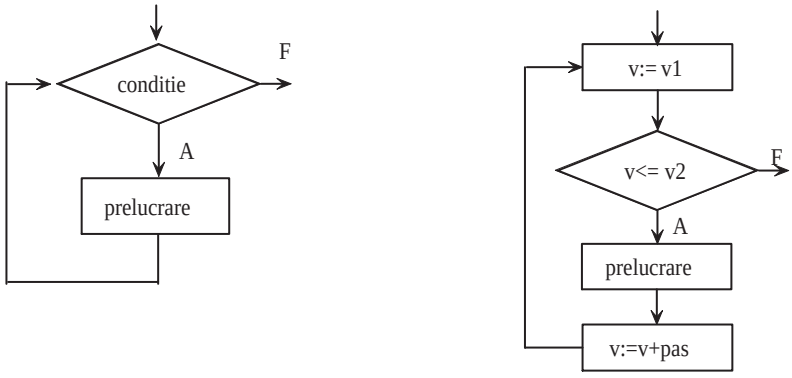

Figura 2: Descrierea grafică a structurii repetitive condiţionată anterior şi a celei cu contor

Un caz particular de structură repetitivă condiţionată anterior este cel al prelucrării repetitive cu contor utilizată atunci când se cunoaşte de la început numărul de repetări ale prelucrării. Se foloseşte o variabilă cu rol de contor a cărei valoare variază între două valori: o valoare iniţială ($v1$) şi o valoare finală ($v2$) fiind modificată la fiecare ciclu cu o altă valoare (pas).

Descrierea

$$\texttt{FOR}\ v \leftarrow \overline{v1, v2, pas}\ \texttt{DO}\ <\text{prelucrare}>$$

este echivalentă, în cazul în care valoarea pas este pozitivă, cu secvenţa:

$$\begin{array}{l} v \leftarrow v1 \\ \texttt{WHILE}\ v \leq v2\ \texttt{DO} \\ \quad \|<\text{prelucrare}> \\ \quad \|v \leftarrow v + pas \end{array}$$

În cazul în care valoarea pas este negativă se schimbă doar condiţia de la `WHILE` (devine $v \geq v2$). În cazul în care $pas = 1$ se va omite din descriere. Descrierea grafică a structurii repetitive cu contor este ilustrată în figura 2.

Varianta condiţionată posterior se specifică prin:

$$\texttt{REPEAT}\ <\text{prelucrare}>\ \texttt{UNTIL}\ <\text{conditie}>$$

La execuţie, prelucrarea se repetă până când condiţia devine adevărată. Prelucrarea se efectuează cel puţin o dată chiar dacă condiţia este de la început adevărată. Dacă în cadrul prelucrării nu se modifică componente ale condiţiei astfel încât aceasta să devină adevărată atunci prelucrarea va fi repetată la nesfârşit. Descrierea grafică a structurii repetitive condiţionată posterior este ilustrată în figura 3.

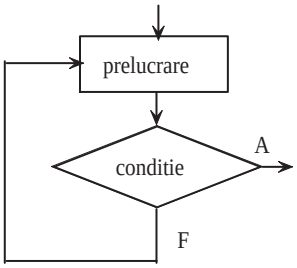

Figura 3: Descrierea grafică a structurii repetitive condiţionată posterior

2 Specificarea datelor

În cadrul algoritmilor uneori se declară variabilele ce vor fi utilizate specificând în acelaşi timp tipul lor. De exemplu, pentru specificarea datelor simple se pot folosi următoarele cuvinte cheie: `INTEGER` (pentru valori întregi), `REAL` (pentru valori reale), `BOOLEAN` (pentru valori logice care pot lua doar valorile `TRUE` - adevărat şi `FALSE` - false) şi `CHAR` (pentru variabile ce pot lua ca valori simboluri oarecare). Astfel, dacă a este o variabilă cu valori întregi, b una cu valori reale, c o variabilă de tip logic iar d una de tip caracter se va specifica:

`INTEGER` a
`REAL` b
`BOOLEAN` c
`CHAR` d

Structura de tip tablou se specifică indicând natura elementelor şi limitele între care variază indicii. De exemplu, în cazul unui tablou unidimensional cu elemente întregi se specifică `INTEGER` $t1[n1..n2]$, iar în cazul unuia bidimensional cu elemente reale şi indici de linie variind între $m1$ şi $m2$ iar indici de coloană variind între $n1$ şi $n2$ se specifică `REAL` $t2[m1..m2, n1..n2]$. Tabloul $t1$ poate corespunde unui şir finit (sau unei mulţimi) cu $n2 - n1 + 1$ elemente iar $t2$ unei matrici cu $m2 - m1 + 1$ linii şi $n2 - n1 + 1$ coloane. Elementele unui tablou se specifică prin intermediul indicilor lor. De exemplu elementul aflat pe poziţia i ($n1 \leq i \leq n2$) în $t1$ se specifică prin $t1[i]$ iar cel aflat pe linia i şi coloana j ($m1 \leq i \leq m2$, $n1 \leq j \leq n2$) în $t2$ se specifică prin $t2[i, j]$. Pot fi specificate şi subtablouri constând din elemente consecutive prin $t1[i1..i2]$ ($n1 \leq i1 < i2 \leq n2$). În cazul în care $n1 > n2$ tabloul $t[n1..n2]$ este vid.

3 Exemple

Structura secvenţială.
Enunţul problemei. Să se calculeze aria unui triunghi cunoscând lungimile laturilor sale (a, b şi c).
Metoda de rezolvare. Se foloseşte formula lui Heron: $aria = \sqrt{p(p-a)(p-b)(p-c)}$ unde p este semiperimetrul.
Descrierea algoritmului.

```
REAL a, b, c, p, aria     { specificarea variabilelor utilizate}
READ a, b, c     {citirea datelor de intrare}
p ← (a + b + c)/2     {calcularea semiperimetrului}
aria ← √(p(p − a)(p − b)(p − c))     {calcularea ariei}
WRITE aria     {afişarea rezultatului }
```

Structura de decizie.

Enunţul problemei. Să se calculeze rădăcinile reale ale ecuaţiei $ax^2 + bx + c = 0$ unde a, b şi c sunt valoari reale arbitrare.

Metoda de rezolvare. Se foloseşte metoda de rezolvare a ecuaţiilor de gradul 2 folosind formulele pentru calculul discriminantului şi pentru calculul rădăcinilor.

Descrierea algoritmului.

```
REAL a, b, c, delta, x1, x2     { specificarea variabilelor utilizate}
READ a, b, c     {citirea datelor de intrare}
delta ← b^2 − 4ac     {calculul discriminantului}
{Analiza semnului discriminantului}
IF delta < 0 THEN WRITE "Ecuatia nu are radacini reale"
ELSE
   ||IF delta = 0 THEN
   ||    ||    ||x1 ← −b/(2a)
   ||    ||    ||WRITE "Rădăcina comună:", x1
   ||    ||ELSE
   ||    ||    ||x1 ← (−b + √delta)/(2a)
   ||    ||    ||x2 ← (−b − √delta)/(2a)
   ||    ||    ||WRITE x1, x2
```

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

Descrierea algoritmilor. (a) O primă variantă de rezolvare bazată pe compararea valorilor două câte două este:

```
REAL a, b, c, min     { specificarea variabilelor utilizate}
READ a, b, c     {citirea datelor de intrare}
IF a < b THEN
        IF a < c THEN min ← a
                 ELSE min ← c
ELSE
        IF b < c THEN min ← b
                 ELSE min ← c
WRITE min     {afişarea rezultatului }
```

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

```
REAL a, b, c, min
READ a, b, c
min ← a     { valoarea iniţială a minimului }
IF b < min THEN min ← b     { actualizarea minimului}
IF c < min THEN min ← c     { reactualizarea minimului}
WRITE min
```

(b) O primă variantă este aceea în care se identifică toate cele şase variante asociind câte o condiţie fiecăreia:

```
REAL a, b, c, min
READ a, b, c
IF a < b AND b < c THEN WRITE a, b, c
IF a < c AND c < b THEN WRITE a, c, b
IF c < a AND a < b THEN WRITE c, a, b
IF c < b AND b < a THEN WRITE c, b, a
IF b < a AND a < c THEN WRITE b, a, c
IF b < c AND c < a THEN WRITE b, c, a
```

O altă variantă în care se compară succesiv câte două valori este:

```
REAL a, b, c, min
READ a, b, c
IF a < b THEN
      IF b < c THEN WRITE a, b, c
            ELSE IF a < c WRITE a, c, b
                  ELSE WRITE c, a, b
ELSE
      IF a < c THEN WRITE b, a, c
            IF b < c THEN WRITE b, c, a
            ELSE WRITE c, b, a
```

Structuri repetitive. La descrierea unei structuri repetitive trebuie stabilite: (i) valorile iniţiale ale variabilelor ce intervin în prelucrarea repetitivă; (ii) prelucrarea/ prelucrările care se repetă; (iii) criteriul de oprire.

Calculul unor sume. Să se calculeze sumele: (a) $\sum_{i=1}^{n} i^3$, $n \in N^*$; (b) $\sum_{i=1}^{n} x^i/i!$ pentru $x \in (0,1)$ şi $n \in N^*$; (c) $\sum_{i=1}^{n(\epsilon)!} x^i/i!$, unde $x \in (0,1)$, $\epsilon > 0$ iar $n(\epsilon) \in N^*$ este cea mai mică valoare naturală pentru care $x^{n(\epsilon)}/n(\epsilon) < \epsilon$ (suma specificată poate fi considerată o aproximare cu precizia ϵ a sumei infinite $\sum_{i=1}^{\infty} x^i/i!$).

Descrierea algoritmilor. (a) Prelucrarea care se repetă este cea de adunare a unui termen. Prelucrarea repetitivă este finalizată în momentul în care au fost adunaţi toţi termenii sumei. Prelucrarea poate fi descrisă utilizând oricare dintre cele trei variante de structură repetitivă. Următoarele trei descrieri sunt echivalente:

```
INTEGER S, i, n
READ n
S ← 0
i ← 1
WHILE i ≤ n
    ‖S ← S + i^3
    ‖i ← i + 1
WRITE S
```

```
INTEGER S, i, n
READ n
S ← 0
FOR i ← 1, n DO S ← S + i^3
WRITE S
```

```
INTEGER S, i, n
READ n
S ← 0
i ← 1
REPEAT
    ‖S ← S + i^3
    ‖i ← i + 1
UNTIL i > n
WRITE S
```

(b) Suma poate fi rescrisă ca $\sum_{i=1}^{n} T(i)$ unde $T(i) = x^i/i!$. Problema ar putea fi rezolvată într-o manieră similară celei de la pct. (a), calculând pentru fiecare $i = \overline{1,n}$ valoarea termenului $T(i)$. Se observă însă că între termenii succesivi există o relaţie care permite calculul lui $T(i)$ pornind de la $T(i-1)$:

$$T(i) = \frac{x^i}{i!} = \frac{x^{i-1}}{(i-1)!} \cdot \frac{x}{i} = T(i-1) \cdot \frac{x}{i}$$

Cum $n \in N^*$ rezultă că suma conţine cel puţin un termen, astel că prelucrarea poate fi descrisă printr-o structură de tip `REPEAT` după cum urmează:

```
INTEGER i, n
REAL x, S
READ x, n
S ← 0
T ← 1
i ← 1
REPEAT
T ← T * x/i     { determinarea valorii termenului curent }
S ← S + T       { adăugarea termenului la sumă }
i ← i + 1       { trecerea la următorul termen}
UNTIL i > n
WRITE S
```

(c) Întrucât nu se cunoaşte numărul de termeni se va folosi un alt criteriu de oprire: prelucrarea se opreşte după ce a fost adăugat primul termen mai mic decât valoarea ϵ (şirul $T(i)$ fiind descrescător, toţi termenii $T(i)$ cu $i > n(\epsilon)$ sunt mai mici decât ϵ). Algoritmul poate fi descris prin:

```
INTEGER i
REAL x, S, ε
READ x, ε
S ← 0
T ← 1
i ← 1
REPEAT
T ← T * x/i     { determinarea valorii termenului curent }
S ← S + T       { adăugarea termenului la sumă }
i ← i + 1       { trecerea la următorul termen}
UNTIL T < ε
WRITE S
```

Determinarea celui mai mare divizor comun. Să se determine cel mai mare divizor comun a două numere naturale nenule, a şi b.

Metoda de rezolvare. Se împarte a la b şi se reţine restul r. Dacă r este nul atunci cel mai mare divizor comun este b. Altfel se împarte b la r şi se reţine din nou restul. Procesul continuă, folosind ca deîmpărţit vechiul împărţitor şi ca împărţitor ultimul rest obţinut, până când se ajunge la un rest nul. Ultimul rest nenul (ultimul împărţitor) reprezintă cel mai mare divizor comun al numerelor a şi b.

Metoda de mai sus, cunoscută sub numele de algoritmul lui Euclid, poate fi descrisă matematic după cum urmează:

$$a = bq_1 + r_1, \quad 0 \leq r_1 < b$$
$$b = r_1q_2 + r_2, \quad 0 \leq r_2 < r_1$$
$$r_1 = r_2q_3 + r_3, \quad 0 \leq r_3 < r_2$$
$$\vdots$$
$$r_{i-2} = r_{i-1}q_i + r_i, \quad 0 \leq r_i < r_{i-1}$$
$$\vdots$$
$$r_{n-1} = r_nq_{n+1} + r_{n+1}, \quad r_{n+1} = 0$$

Metoda are caracter algoritmic deoarece succesiunea de împărţiri este finită (şirul resturilor un şir strict descrescător de valori naturale).

Descrierea algoritmului. Prelucrarea este cu caracter repetitiv. Varianta condiţionată anterior poate fi descrisă prin:

```
INTEGER a, b, d, i, r
READ a, b
d ← a      {iniţializarea deîmpărţitului}
i ← b      {iniţializarea împărţitorului}
r ← dMODi      {calculul restului}
WHILE r ≠ 0 DO
    ‖d ← i      {noul deîmpărţit}
    ‖i ← r      {noul împărţitor}
    ‖r ← dMODi      {noul rest}
WHILE i      {afişarea ultimului rest nenul }
```

Varianta cu prelucrare repetitivă condiţionată posterior este:

```
INTEGER a, b, d, i, r
READ a, b
d ← a     {iniţializarea deîmpărţitului}
i ← b     {iniţializarea împărţitorului}
REPEAT
    ||r ← d MOD i     {calculul restului}
    ||d ← i     {noul deîmpărţit}
    ||i ← r     {noul împărţitor}
UNTIL r = 0
WRITE d     {afişarea ultimului rest nenul }
```

Calculul mediei aritmetice. Să se calculeze media aritmetică a elementelor unui şir finit de n numere reale: $(x_1, x_2, \ldots, x_n)$.

Metoda de rezolvare. Media aritmetică se obţine cu formula $(\sum_{i=1}^{n} x_i)/n$.

Descrierea algoritmului. Şirul poate fi reprezentat printr-un tablou de forma $x[1..n]$ iar prelucrarea este una repetitivă pentru care se cunoaşte numărul de repetări. Descrierea cea mai simplă este cea care foloseşte structura repetitivă cu contor:

```
REAL x[1..n], medie
INTEGER n,i
READ x[1..n]     { citirea elementelor şirului }
medie ← 0     { iniţializarea variabilei ce va conţine valoarea mediei }
FOR i ← 1, n DO     {calculul sumei elementelor}
    ||medie ← medie + x[i]     { adăugarea câte unui termen}
medie = medie/n     {împarţirea sumei la numărul elementelor}
WRITE medie
```

4 Tehnica rafinării succesive şi subalgoritmi

Cea mai simplă metodă de rezolvare algoritmică a problemelor este de a le descompune în subprobleme şi de a le rezolva pe acestea din urmă. Rezolvarea fiecărui tip de subproblemă se va realiza în cadrul unui (sub)algoritm, iar algoritmul de rezolvare a problemei iniţiale va utiliza subalgoritmii pentru a construi rezultatul final. Dacă problema conţine mai multe subprobleme de aceeaşi natură atunci acestea pot fi rezolvate de acelaşi subalgoritm. În acest scop prelucrările efectuate în subalgoritm vor fi efectuate asupra unor *date generice* ce vor fi înlocuite cu datele concrete specifice problemei doar în momentul execuţiei. Transferul controlului execuţiei de la algoritm la un subalgoritm se numeşte *apel*, datele generice sunt numite *parametri formali*, iar valorile lor concrete sunt numite *parametri efectivi*. Efectul unui subalgoritm constă fie în *returnarea* unei valori către algoritm fie în modificarea, prin intermediul parametrilor, a unora dintre variabilele algoritmului.

Structura generală a unui subalgoritm este:

```
<nume subalgoritm> (< date generice >)
    ||< date ajutătoare >
    ||< prelucrări specifice >
    ||RETURN < rezultate >     {returnarea rezultatelor}
```

Uneori un subalgoritm poate avea, pe lângă rezultatele pe care le returneaza, şi *efecte laterale*. Acestea constau de regulă în modificarea valorilor parametrilor sau a altor date aparţinând algoritmului general.

Exemplul 1. *Enunţul problemei.* Să se determine cel mai mare dintre minimele valorilor de pe fiecare linie a unei matrici $(a_{ij})_{i=\overline{1,m},j=\overline{1,n}}$ $(\max_{i=\overline{1,m}} \min_{j=\overline{1,n}} a_{ij})$.
Metoda de rezolvare. Se construieşte un vector ce conţine valorile minime de pe linii: $b_i = \min_{j=\overline{1,n}} a_{ij}$, $i = \overline{1,m}$ după care se determină valoarea maximă din acest vector. Problema presupune astfel rezolvarea a două subprobleme: determinarea valorii minime dintr-un şir finit cu n elemente şi determinarea valorii maxime dintr-un şir finit cu m elemente.
Descrierea subalgoritmilor. Determinarea minimului unui şir finit reprezentat printr-un tablou cu n elemente reale se bazează pe compararea valorii unei variabile iniţializată cu primul element al şirului cu fiecare dintre următoarele elemente. În cazul în care este întâlnită o valoare mai mică atunci aceasta este reţinută în variabilă. Această prelucrare repetitivă poate fi descrisă prin:

```
minim (REAL x[1..n])
REAL min     { variabilă în care se va reţine valoarea minimului }
INTEGER i    { contor ce va fi folosit pentru parcurgerea şirului }
min ← x[1]   { iniţializarea valorii minimului }
FOR i ← 2, n DO     { parcurgerea tabloului }
   ||IF min > x[i] THEN min ← x[i]     {înlocuirea valorii minimului dacă este cazul}
RETURN min     { returnarea rezultatului }
```

Subalgoritmul ce permite determinarea maximului unui şir cu m elemente poate fi descris prin:

```
maxim (REAL y[1..m])
REAL max     { variabilă în care se va reţine valoarea maximului }
INTEGER i    { contor ce va fi folosit pentru parcurgerea şirului }
max ← y[1]   { iniţializarea valorii maximului }
FOR i ← 2, n DO     { parcurgerea tabloului }
   ||IF max < y[i] THEN max ← x[i]     {înlocuirea valorii maximului dacă este cazul}
RETURN max     { returnarea rezultatului }
```

Descrierea algoritmului. Algoritmul constă în construirea (folosind subalgoritmul `minim`) elementelor unui tablou b şi în determinarea (folosind subalgoritmul `maxim`) valorii maxime din acest tablou.

```
REAL a[1..m, 1..n]     {matricea }
REAL b[1..m]     {tabloul ce conţine valorile minime de pe liniile matricii }
REAL c     { variabila în care se reţine rezultatul}
INTEGER m, n, i
READ a[1..m, 1..n]     {citirea matricii}
FOR i ← 2, n DO     {construirea tabloului cu minime}
   ||b[i] ← minim(a[i, 1..n])     {determinarea minimului de pe linia i }
c ← maxim(b[1..m])     {determinarea maximului din tabloul b }
WRITE c     {afişarea rezultatului}
```

Exemplul 2. *Descrierea problemei.* Se consideră un număr natural constituit din 10 cifre distincte (de exemplu, 6709385421). Să se determine numărul imediat următor (în ordine crescătoare) din şirul tuturor numerelor naturale constituite din 10 cifre distincte.

Metoda de rezolvare. Considerăm numărul reprezentat prin tabloul cifrelor sale, $x[1..10]$, în care cea mai semnificativă se află pe prima poziţie. Atâta timp cât cifrele numărului nu sunt în ordine descrescătoare (adică 9876543210) poate fi construit numărul cerut parcurgând următoarele etape:

Pas 1. Se parcurge şirul de la dreapta la stânga şi se identifică prima pereche (x_i, x_{i-1}) cu proprietatea $x_i > x_{i-1}$. Dacă numărul iniţial nu este 9876543210, atunci există o astfel de pereche.

Pas 2. Se determină

$$x_k = \min\{x_j | j = \overline{i,n} \text{ cu } x_j > x_{i-1}\}$$

adică cel mai mic element al subtabloului $x[i..n]$ care îl depăşeşte pe x_{i-1} (existenţa unui astfel de element este asigurată de proprietatea $x_i > x_{i-1}$).

Pas 3. Se interschimbă x_{i-1} cu x_k. În felul acesta se obţine un număr mai mare decât cel iniţial.

Pas 4. Se ordonează crescător subtabloul $x[i..n]$. Scopul ordonării crescătoare este acela de a obţine numărul imediat următor celui iniţial care satisface cerinţa de a fi constituit din cifre distincte.

Descrierea algoritmului. Algoritmul poate fi descompus în următorii subalgoritmi, corespunzători principalelor prelucrări specificate în descrierea de mai sus:

- `Identificare`: pentru găsirea perechii (x_{i-1}, x_i) cu proprietatea $x_{i-1} < x_i$. Subalgoritmul primeşte ca parametru întreg tabloul x şi returnează valoarea i. Dacă $i = 1$ rezultă că nu există succesorul cerut pentru numărul iniţial.
- `Minim`: pentru determinarea valorii minime din subtabloul $x[i..n]$ care are proprietatea că este mai mare decât x_{i-1}. Subalgoritmul va primi ca parametri: tabloul $x[1..n]$ şi indicele i şi va returna indicele valorii minime.
- `Sortare`: pentru ordonarea crescătoare a subtabloului $x[i..n]$

Structura generală a algoritmului este:

```
INTEGER x[1..n], i, k
READ x[1..n]
i ←Identificare(x[1..n])
IF i = 1 THEN WRITE "Nu exista"
ELSE
    ||k ←Minim(x[1..n], i)
    ||x[i − 1] ↔ x[k]
    ||Sortare(x[i..n])
    ||WRITE x[1..n]
```

Subalgoritmii destinaţi identificării perechii (x_{i-1}, x_i) şi a indicelui minimului pot fi descrişi după cum urmează:

```
Identificare (x[1..n])
INTEGER i
i ← n
WHILE (x[i] < x[i − 1])AND(i > 1) DO i ← i − 1
RETURN i
```

respectiv

```
Minim (x[1..n], i)
INTEGER j
k ← i
FOR j ← i + 1, n DO
   IF x[j] < x[k]AND  x[j] > x[i − 1] THEN k ← j
RETURN k
```

În continuare majoritatea algoritmilor vor fi descrişi sub forma unor subalgoritmi, iar pentru simplificare natura datelor generice şi a celor locale ajutătoare nu va mai fi specificată.

5 Exerciţii.

Descrieţi în limbaj algoritmic metodele de rezolvare a următoarelor probleme:

1. Să se calculeze: (a) $\prod_{i=1}^{n}(i+1)$; (b) $\sum_{i=1}^{n} 1/(i!)^2$.

2. Să se aproximeze, cu precizia ϵ sumele: (a) $\sum_{i=1}^{\infty} x^i/i!$; (b) $\sum_{i=0}^{\infty}(-1)^i x^{2i}/(2i)!$; (c) $\sum_{i=0}^{\infty}(-1)^i x^{2i+1}/(2i+1)!$; (d) $\sum_{i=1}^{\infty}(-1)^{i+1} x^i/i$.

3. Să se genereze primele N elemente şi să se aproximeze cu precizia ϵ limitele şirurilor: (a) $x_n = (1+1/n)^n$; (b) $x_1 = a$, $x_n = (x_{n-1} + a/x_{n-1})/2$ $(a > 0)$; (c)$x_n = f_{n+1}/f_n$, $f_1 = f_2 = 1$, $f_n = f_{n-1} + f_{n-2}$, $n > 2$.

4. (a) Să se determine toţi divizorii unui număr natural; (b) Să se determine toţi divizorii comuni a două numere naturale; (c) Să se verifice dacă un număr este prim sau nu; (d) Să se verifice dacă două numere sunt prime între ele; (e) Să se genereze toate numerele prime mai mici decât o valoare dată; (f) Să se genereze toate perechile de numere prime între ele mai mici decât o valoare specificată; (g) Să se descompună în factori primi un număr natural; (h) Să se determine cel mai mare divizor comun şi cel mai mic multiplu comun folosind descompunerea în factori primi;

5. (a) Să se determine suma tuturor cifrelor unui număr natural; (b) Să se determine toate cifrele distincte dintr-un număr natural; (c) Să se determine valoarea obţinută prin inversarea cifrelor unui număr natural; (d) Să se determine reprezentarea în baza doi a unui număr natural.