LIMBAJUL PSEUDOCOD

Introducere

Pseudocod este un limbaj prin care sunt descriși pașii dintr-un algoritm. Limbajul pseudocod conține structurile specifice unui limbaj de programare obișnuit (precum Pascal, C/C++, Basic, etc) dar este destinat a fi citit de către oameni, nu de către calculatoare.
De obicei sunt omise detaliile vitale într-un limbaj de programare, precum declararea variabilelor sau secvențe specifice limbajului. Algoritmii pseudocod pot conține secvențe descrise în limbaj natural, precum și expresii matematice compacte, care lipsesc din limbajele de programare reale.
Conceptele care intervin in algoritmii pseudocod sunt similare cu cele din limbajele de programare:

- instrucțiuni
- variabile
- constante
- operații
- expresii

Există mai multe variante ale limbajului pseudocod. Acest articol se referă la varianta folosită în subiectele pentru examenul de bacalaureat la informatică. Limba folosită este limba română.

Variabile și constante

În pseudocod variabilele nu se declară, iar numele lor respectă regula uzuală a identificatorilor – litere, cifre, caracterul de subliniere și să nu înceapă cu cifră.
Constantele care apar de regulă în algoritmii pseudocod sunt constante literale (valori numerice, întregi sau reale, caractere – delimitate prin apostrof ', șiruri de caractere – delimitate prin apostroafe (') sau ghilimele ("). De asemenea, pot să apară constante matematice, de exemplu π.

Operații

Sunt prezente toate operațiile aritmetice uzuale:

- adunarea a două numere: `a+b`. În funcție de context, rezultatul este întreg sau real.
- scăderea a două numere: `a-b`. În funcție de context, rezultatul este întreg sau real.
- înmulțirea a două numere: `a*b`. În funcție de context, rezultatul este întreg sau real.
- împărțirea a două numere: `a/b`. Rezultatul se consideră real. Dacă se dorește determinarea câtului împărțirii a două numere naturale, se va folosi partea întreagă a rezultatului împărțirii: `[a/b]`.
- restul împărțirii a două numere întregi: `a%b`. Se aplică numai pentru date întregi, iar rezultatul este număr întreg.

Notă: Unele variante pseudocod propun operațiile `DIV` și `MOD` pentru determinarea câtului și restului împărțirii a două numere întregi. De asemenea, pot să apară operații de ridicare la putere, în forma $a^n a^n$, sau de extragere a rădăcinii pătrate – $x--\sqrt{x}$.

Operațiile relaționale sunt:

- $=$ – egalitatea
- $\neq$ – neegalitatea
- $<$ – mai mic
- $\leq$ – mai mic sau egal
- $>$ – mai mare
- $\geq$ – mai mare sau egal

Operanzii sunt de regulă numere, iar rezultatul este valoare de adevăr (`adevărat` sau `fals`). Operațiile logice sunt `NOT`, `ȘI`, `SAU` – cu semnificațiile cunoscute. O descriere a operațiilor logice poate fi găsită aici: www.pbinfo.ro/articole/21315/operatii-logice.

Instrucțiuni

Instrucțiunile sunt componentele algoritmului care au efect, atunci când se execută. Ele modifică valorile unor variabile, citesc sau afișează date, repetă anumite acțiuni, etc.
De regulă fiecare instrucțiune se scrie pe o linie (sau mai multe în cazul celor complexe), dar există situații când, pentru a economisi spațiu, două sau mai multe instrucțiuni simple se scriu pe același rând, separate prin **;**.
Orice algoritm poate fi reprezentat prin intermediul a trei tipuri de structuri:

- structura liniară
- structura alternativă
- structura repetitivă
  - structura repetitivă cu număr necunoscut de pași
    - structura repetitivă cu număr necunoscut de pași și test inițial
    - structura repetitivă cu număr necunoscut de pași și test final
  - structura repetitivă cu număr cunoscut de pași

Citirea

Pentru citirea datelor se folosește instrucțiunea `citește <listă variabile>`, unde `<listă variabile>` reprezintă un șir de variabile separate prin caracterul **,**. Se preiau valori succesive de la tastatură și se memorează în variabilele din listă.
Exemplu

```
citeste a , b , c
```

Citirea datelor este frecvent însoțită de precizări privind datele citite (tip, valori posibile. etc.).

Afișarea

Pentru afișarea datelor se folosește instrucțiunea `scrie <listă expresii>`, unde `<listă expresii>` reprezintă un șir de expresii separate prin caracterul **,**. Se evaluează în ordine expresiile din listă și se afisează pe ecran rezultatele lor.
Exemplu

```
scrie 'a + b = ' , a + b
```

Dacă `a = 3` și `b = 4` se va afișa:

```
a + b = 7
```

Atribuirea

Pentru atribuire se folosește instructiunea `<variabilă> ← <expresie>`. Se evaluază expresia, iar valoarea obținută se memorează in variabilă.
Exemple

```
        a ← 0
S ← a + b
i ← i + 1
```

Structura alternativă

În pseudocod există instrucțiunea `daca`, cu următoarea sintaxă:

```
daca <conditie> atunci
    <instructiuni1>
altfel
    <instructiuni2>
```

sau

```
daca <conditie> atunci
    <instructiuni1>
```

Modul de execuție este următorul:

- se evaluează `<conditie>`
- dacă este adevărată, se executa grupul de instrucțiuni `<instructiuni1>`, apoi se trece la următoarea instrucțiune
- dacă este falsă, se executa grupul de instrucțiuni `<instructiuni2>`, dacă există secțiunea `altfel`, apoi se trece la următoarea instrucțiune.

Structuri repetive

Structura repetitivă cu număr cunoscut de pași

Instrucțiunea PENTRU

Sintaxă

```
pentru <variabila> ← <expresie initiala>, <expresie finala> , <pas> executa
    <instructiuni>
```

unde `expresie initiala`, `expresie finala` și `pas` sunt expresii cu rezultat (de regulă) întreg, iar `pas` poate fi `1` sau `-1` și poate lipsi, caz în care se consideră că este `1`.
`variabila` primește pe rând valori crescătoare (dacă `pas = 1`) sau descrescătoare (dacă `pas = -1`) începând de la `expresie initiala` până la `expresie finala`, și pentru fiecare valoare se execută secvența `instructiuni`.
Dacă `pas = 1` și `expresie initiala > expresie finala>`, secvența `instructiuni` nu se va executa deloc. La fel se întâmplă când `<pas = -1` și `expresie initiala < expresie finala`. În caz contrar numărul de pași este `expresie initială - expresie finala + 1`, dacă `<pas = 1`, respectiv `expresie finală - expresie initiala + 1`, dacă `pas=-1`.

Exemplu

Determinarea sumei și produsului primelor `n` numere naturale:

```
S ← 0; P ← 1
pentru i←1,n executa
    S ← S + i
    P ← P * i
scrie S, ' ' , P
```

Structura repetitivă cu număr necunoscut de pasi și test inițial

Instrucțiunea CÂT TIMP … EXECUTĂ

Sintaxă

```
cat timp <conditie> executa
    <instructiuni>
```

Mod de execuție

1. se evaluează condiția
2. dacă este adevărată se executa instrucțiunile și se revine la pasul 1
3. dacă este falsă se trece la următoarea instrucțiune din algoritm

Important: Dacă condiția este de la început falsă, instrucțiunile nu se vor executa deloc!

Exemplu

Determinarea sumei cifrelor unui număr natural:

```
citeste n
S ← 0
cat timp n ≠ 0 executa
    S ← S + n % 10
    n ← [n/10]
scrie S
```

Structura repetitivă cu număr necunoscut de pasi și test final

Instrucțiunea EXECUTA … CAT TIMP

Sintaxă

```
executa
    <instructiuni>
cat timp <conditie>
```

Mod de execuție

1. se executa instrucțiunile
2. se evaluează condiția
3. dacă este **adevărată** se revine la pasul 1
4. dacă este **falsă** se trece la următoarea instrucțiune din algoritm

Important: Chiar dacă condiția este de la început falsă, instrucțiunile s-au executat deja o dată!

Exemplu

Numărul cifrelor unui număr natural:

```
citeste n
cnt ← 0
executa
    cnt ← cnt + 1
    n ← [n/10]
cat timp n ≠ 0
scrie cnt
```

Instrucțiunea REPETA … PANA CAND

Sintaxă

```
repeta
    <instructiuni>
pana cand <conditie>
```

Mod de execuție

1. se executa instrucțiunile
2. se evaluează condiția
3. dacă este **falsă** se revine la pasul 1
4. dacă este **adevărată** se trece la următoarea instrucțiune din algoritm

Important: Chiar dacă condiția este de la început adevărată, instrucțiunile s-au executat deja o dată!

Exemplu

Numărul cifrelor unui număr natural:

```
citeste n
cnt ← 0
repeta
    cnt ← cnt + 1
    n ← [n/10]
pana cand n = 0
scrie cnt
```

Echivalența structurilor repetitive

Numeroase exerciții propun un algoritm și se cere scrierea unui algoritm echivalent care să folosească o structură repetitivă de alt tip. Doi algoritmi sunt echivalenți dacă pentru orice set de date de intrare (conforme cu restricțiile problemei) ei obțin aceleași rezultate.
Această secțiune descrie câteva reguli de transformare a structurilor repetitive din algoritmii pseudocod.

PENTRU → CÂTTIMP

```
pentru i←a,b executa
    <instructiuni>
```

```
i ← a
cattimp i ≤ b executa
    <instructiuni>
    i ← i + 1
```

PENTRU → EXECUTĂ … CÂTTIMP

```
pentru i←a,b executa
    <instructiuni>
```

```
i ← a
daca i ≤ b atunci
    executa
        <instructiuni>
        i ← i + 1
    cattimp i ≤ b
```

PENTRU → REPETĂ … PÂNÂCÂND

```
pentru i←a,b executa
    <instructiuni>
```

```
i ← a
daca i ≤ b atunci
    repeta
        <instructiuni>
        i ← i + 1
    panacand i > b
```

CÂTTIMP → EXECUTĂ … CÂTTIMP

```
cattimp <conditie> executa
    <instructiuni>
```

```
daca <conditie> atunci
    executa
        <instructiuni>
    cattimp <conditie>
```

REPETĂ … PÂNĂCÂND → CÂTTIMP

```
repeta
    <instructiuni>
panacand <conditie>
```

```
<instructiuni>
cattimp NOT <conditie> executa
    <instructiuni>
```