Pseudocod

Este un limbaj mai apropiat de cel natural, utilizat pt. a descrie algoritmi.

Cel utilizat la Bac are următoarea formă:

C++	Pseudocod
1. Citirea cin >> v1 >> v2 >> ... ; 2. Afișarea cout << e1 << e2 << ... ; 3. Atribuirea variabila = expresie ;	citește v1, v2, ... scrie e1, e2, ... variabila ← expresie sau :=

4A)

```
if(cond)
    secvență
```

```
dacă cond atunci
    secvență
```

4B)

```
if(cond)
    secv-adevăr;
else
    secv-fals;
```

```
dacă cond atunci
    secv_adevăr
altfel
    secv-fals
```

Ex. Transcrierea în C++ a următoarei secvențe

```
dacă a ≠ 0 atunci
    x ← 1/a
    y ← x * x
altfel
    x ← 2
    y ← x * x - 5
```

```
if(a)
{   x = 1/a;
    y = x * x;
}
else
{   x = 2;
    y = x * x - 5;
}
```

5) Repetitivă cu test anterior

```
while (cond)
    secvență
```

Cât timp cond execută
secvență

6) Repetitivă cu test final

```
do
    secvență
while (cond);
```

forma standard:

repetă
secvență
până când negare condiție

forma acceptată

execută
secvență
cât timp condiție

7A) Repetitiva cu contor - crescător

```
for(cnt = li; cnt <= lf; cnt++)
    secvență
```

pentru cnt ← li, lf execută
    secvență

7B) descrescător

```
for(cnt = li; cnt >= lf; cnt--)
    secvență
```

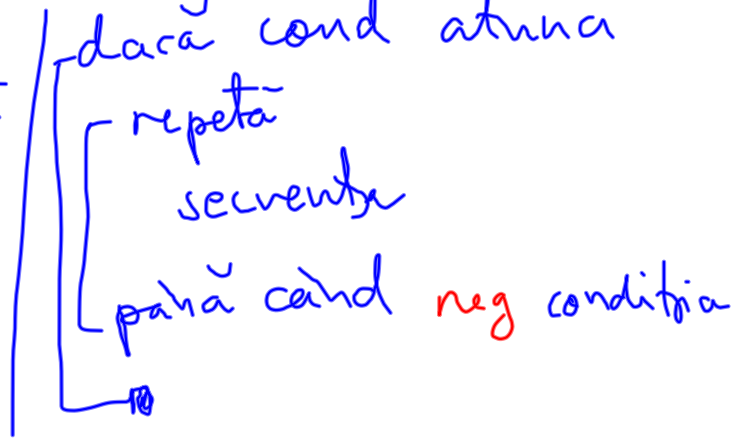
pentru cnt ← li, lf, -1 execută
    secvență

Obs: În pseudocod NU avem voie (spre deosebire de (++) să modificăm cnt, li sau lf în interiorul for-ului.

Reguli de transformare între repetitive a.î. să fie echivalente

1) De pe cât timp (cu test inițial) pe repeta .. până când (cu test final)

cât timp condiție execută
    secvență



2) De pe repetitivă cu test final pe repetitivă cu test inițial:

repetă
  secvență
până când condiție

secvență
cât timp neg condiție execută
  secvență

3) De pe for pe cât timp:

pentru cnt ← li, lf execută
  secvență

cnt ← li
cât timp cnt ≤ lf execută
  secvența
  cnt ← cnt + 1

4) De pe for pe repetă .. până când

```
pentru cnt ← li, lf execută
    secvență
```

```
cnt ← li
dacă li ≤ lf atunci
    repetă
        secvență
        cnt ← cnt + 1
    până când cnt > lf
```

Orice alt fel de transformare NU are reguli, ci trebuie analizată, bazându-se pe o particularitate, artificiu, etc.

Obs: În pseudocod, / se consideră a avea rezultat real. Din acest motiv, pentru câtul împărțirii folosim [a / b]

Ex.:

```
citește a, b   (∈ ℕ*)
k ← 0
cât timp a > b
    k ← k + 1
    a ← a - b
scrie a, k
```

Să se rescrie acest algoritm cu unul echivalent dare care NU utilizează repetitive.

Rezolvare: trebuie să ne dăm seama că secvența calculează câtul și restul prin împărțiri repetate.

Răsp: citește a, b
scrie a%b, [a/b]

Obs.: În pseudocod NU se specifică de regulă detalii despre forma exactă a afișării. Chiar dacă un "scrie" afișează două numere cu virgulă, le considerăm separate (le vom scrie cu spațiu)

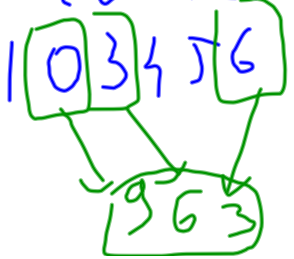
```
citeşte n (număr natural)
z←0
p←1
┌cât timp n>0 execută
| c←n%10
| n←[n/10]
| ┌dacă c%3=0 atunci
| |   z←z+p*(9-c)
| |   p←p*10
| └■
└■
scrie z
```

a) n = 103456

n	z	p	c
~~103456~~	0	~~1~~	~~6~~
~~10345~~	~~3~~	~~10~~	~~5~~
~~1034~~	~~63~~	~~100~~	~~4~~
~~103~~	963	1000	~~3~~
~~10~~			~~0~~
~~1~~			1
0			

Răsp: 963

Algoritmul formează un număr plecând de la cifrele divizibile cu 3 ale celui dat, scrise mai din 9 şi păstrate în aceeaşi ordine.



b) Scrieți toate numere naturale **impare**, distincte, fiecare având **exact** două cifre, care pot fi citite pentru variabila `n` astfel încât să se afișeze valoarea 3. **(4p.)**

Răspuns: 61, 65 și 67.

```
citește n (număr natural)
z←0
p←1
┌cât timp n>0 execută
| c←n%10
| n←[n/10]
| ┌dacă c%3=0 atunci
| |   z←z+p*(9-c)
| |   p←p*10
| |_■
|_■
scrie z
```

c) Scrieți în pseudocod un algoritm, echivalent cu cel dat, în care să se înlocuiască structura `cât timp...execută` cu o structură repetitivă de alt tip. **(6p.)**

```
citește n
z ← 0
p ← 1
dacă n>0 atunci
    execută
        c ← n%10
        n ← [n/10]
        dacă c%3=0 atunci
            z ← z+p*(9-c)
            p ← p*10
    cât timp n>0
scrie z
```

```
citeşte n (număr natural)
z←0
p←1
┌cât timp n>0 execută
| c←n%10
| n←[n/10]
| ┌dacă c%3=0 atunci
| |   z←z+p*(9-c)
| |   p←p*10
| |_
|_
scrie z
```

d)
```
#include <iostream>
using namespace std;
int main()
{ int n,z=0,p=1,c;
  cin >> n;
  while(n>0)
  { c = n%10;
    n/=10;
    if(c%3==0)
    { z = z+p*(9-c);
      p*=10;
    }
  }
  cout << z;
  return 0;
}
```