**Transformări echivalente între structuri repetitive**

Între **anumite** tipuri de instrucțiuni repetitive putem avea "rescrieri" perfect echivalente, care pot fi aplicate în orice tip de situație.

Le vom prezenta în continuare (sunt 4 tipuri).

Din păcate, orice rescriere care NU se află printre ele, necesită o analiză atentă şi de multe ori se bazează pe un artificiu, formulă de calcul, pe deducerea unui fapt realizat de algoritm, etc.

1) De pe repetitivă cu test inițial pe repetitivă cu test final

Se dă:

┌cât timp ***condiție*** execută

│ instrucțiuni

└■

Transformarea sa:

┌dacă ***condiție*** atunci

│ ┌repetă

│ │ instrucțiuni

│ └până când ***not condiție***

└■

Obs: În C++ NU există instrucțiune adaptată după "repetă..până când" (spre exemplu, în Pascal există repeat..until, în Basic există do..loop until). Din acest motiv, se acceptă şi scrierea în Pseudocod de tipul execută ... cât timp. Prin urmare, unui programator de C++ îi va veni mult mai uşor să transcrie astfel:

┌dacă ***condiție*** atunci

│ ┌execută

│ │ instrucțiuni

│ └cât timp ***condiție***

└■

Exemplu:

Să se rescrie următorul pseudocod utilizând o structură repetitivă cu test final

Iată un exemplu de rulare:

Dacă citim x=4184 se va afişa 4

Dacă citim x=217986 se va afişa 2

Dacă citim x=7 se va afişa 7

Dacă citim x=4 se va afişa 4

citeşte x

pc ← x

┌cât timp pc>9 execută

│ pc ← [pc/10]

└■

scrie pc

Iată rescrierea atât pe "repetă..până când":

citeşte x

pc ← x

┌daca pc>9 atunci

│ ┌repetă

│ │ pc ← [pc/10]

│ └până când pc<=9

└■

scrie pc

... cât şi pe "cât timp..execută":

citeşte x

pc ← x

┌dacă pc>9 atunci

│ ┌execută

│ │ pc ← [pc/10]

│ └cât timp pc>9

└■

scrie pc

Obs: Există o serie de cazuri în care acel "dacă" ce îmbracă repetitiva rescrisă NU mai este necesar. Totuşi, păstrarea sa în scriere, conform "rețetei" de mai sus, NU este greşită, este DOAR inutilă.

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

Se dă:

┌repetă

│ instrucțiuni

└până când ***condiție***

Transformarea sa:

instrucțiuni

┌cât timp **not *condiție*** execută

│ instrucțiuni

└■

Obs: Partea mai anevoioasă la acest tip de transformare poate consta în faptul că, dacă instrucțiunile din corpul repetitivei sunt multe, rescrierea lor necesită mai mult spațiu.

O serie de repetitive pot fi scrise şi mai scurt, însă acest lucru depinde de la caz la caz, neavând o rețetă clară, ci depinde de ingeniozitatea programatorului.

Exemplu:

Să se rescrie următorul pseudocod utilizând o structură repetitivă cu test inițial:

citeşte n

Iată un exemplu de rulare:

Dacă citim n=3185 se va afişa 4

Dacă citim n=7 se va afişa 1

Dacă citim n=0 se va afişa 1

nc ← 0

┌repetă

│ n ← [n/10]

│ nc ← nc+1

└până când n=0

scrie nc

citeşte n

nc ← 0

n ← [n/10]

nc ← nc+1

┌cât timp n≠0 execută

│ n ← [n/10]

│ nc ← nc+1

└■

scrie nc

3) De pe repetitivă cu contor pe repetitivă cu test inițial

3b. For descrescător:

┌pentru **contor** ← **li**,**lf,-1** execută

│ **instrucţiuni**

└■

Transformarea sa:

**contor** ← **li**

┌cât timp **contor** >= **lf** execută

│ **instrucţiuni**

│ **contor** ← **contor - 1**

└■

Se dă:

3a. For crescător:

┌pentru **contor** ← **li**,**lf** execută

│ **instrucțiuni**

└■

Transformarea sa:

**contor** ← **li**

┌cât timp **contor** <= **lf** execută

│ **instrucțiuni**

│ **contor** ← **contor + 1**

└■

4) De pe repetitivă cu contor pe repetitivă cu test final

Ne putem gândi că aplicăm 3) şi apoi 1)

3b. For descrescător:

┌pentru **contor** ← **li**,**lf,-1** execută

│ **instrucţiuni**

└■

Transformarea sa:

**contor** ← **li**

┌dacă **contor** >= **lf** atunci

│┌execută

││ **instrucţiuni**

││ **contor** ← **contor - 1**

│└cât timp **contor** >= **lf**

└■

Se dă:

4a. For crescător:

┌pentru **contor** ← **li**,**lf** execută

│ **instrucțiuni**

└■

Transformarea sa:

**contor** ← **li**

┌dacă **contor** <= **lf** atunci

│┌execută

││ **instrucțiuni**

││ **contor** ← **contor + 1**

│└cât timp **contor** <= **lf**

└■

**Transcrierea în C++**

Scopul de bază al limbajului pseudocod este să descrie într-un limbaj natural, relaxat, fără reguli stricte de sintaxă (de ex. noțiunea de diferit se poate scrie ca <>, !=, ≠, sau atribuirea poate fi scrisă şi cu variabilă := valoare; în loc de ←, etc) un anumit algoritm.

O primă abilitate pe care trebuie s-o deprindă un elev care studiază informatica este să transcrie un astfel de limbaj într-un limbaj practic de programare, cu reguli foarte stricte.

În C++ structura unui program este următoarea (aceste lucruri NU fac parte din programul pseudocod, ele trebuie scrise în orice program C++):

#include<iostream>

using namespace std;

int main()

{

declarații de variabile;

instrucțiuni program;

return 0;

}

În cazul mediului de programare CodeBlocks, acest program este generat automat.

În cazul programelor de nivel bac, rolul nostru este să personalizăm acest program.