INFORMATICĂ · CLASA A IX-A · STRUCTURI DE CONTROL · LECȚIA 5

Structuri repetitive (1): while și do-while

Durată: 60 min Nivel: toate profilurile Înainte: lecțiile ix-04

CE VEI ȘTI DUPĂ LECȚIE

- Scrii bucle while (test înainte) și do-while (test după) și știi când folosești fiecare variantă.
- Citești un șir de numere până la apariția lui 0 (sau a altei valori de oprire).
- Calculezi sume și numărări „din mers", fără să memorezi toate valorile.
- Recunoști și eviți buclele infinite.
- Transformi o structură cât timp într-una repetă...până când (și invers), ca la Bac.

1. De ce avem nevoie de repetiție?

„Citește numere până la 0 și afișează-le suma." Nu știm câte numere vor fi, deci nu putem scrie de mână câte un `cin` pentru fiecare. Avem nevoie de o instrucțiune care **repetă** un bloc de câte ori e nevoie.

Imaginează-ți o casieră: ia produs după produs și adună prețul la total, **până când** nu mai sunt produse. Nu știe de la început câte sunt; știe doar ce face la fiecare pas și când se oprește.

2. `while` — testul înainte

```
while (conditie)
{
    // corpul buclei: se repetă cât timp condiția e adevărată
}
```

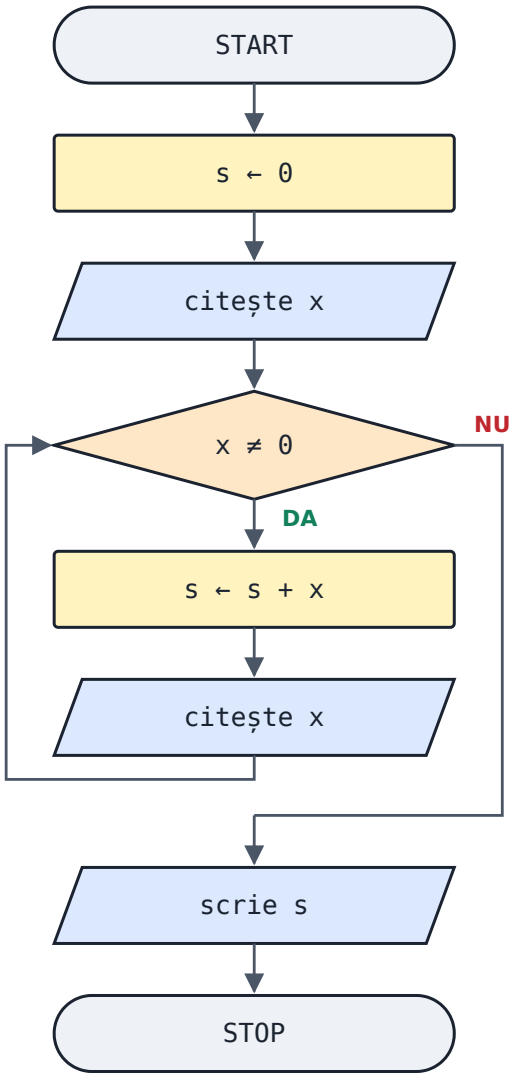
Pseudocod ↔ C++:

```
┌cât timp n ≠ 0 execută          while (n != 0)
│   s ← s + n % 10              {   s = s + n % 10;
│   n ← [n / 10]                    n = n / 10;
└■                               }
```

La fiecare **iterație** (repetare) calculatorul face: (1) testează condiția; (2) dacă e adevărată, execută corpul și se întoarce la (1); (3) dacă e falsă, sare după buclă.

- Dacă de la început condiția e falsă, corpul **nu se execută deloc**.
- În corp trebuie să se modifice ceva din condiție, altfel bucla nu se oprește.

Șablonul „citire până la 0”

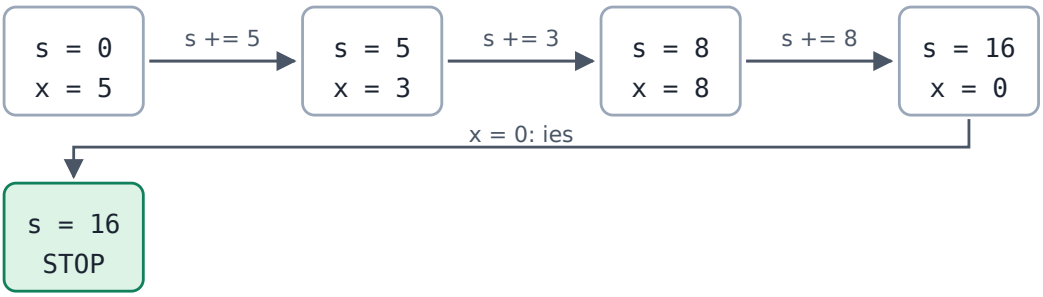


Citim înainte de buclă și la sfârșitul corpului

Șablonul citire până la 0

```cpp
#include <iostream>
using namespace std;

int main()
{
    int x, s = 0;
    cin >> x;                // citim PRIMA valoare înainte de buclă
    while (x != 0)
    {
        s = s + x;           // prelucrăm valoarea curentă
        cin >> x;            // citim URMĂTOAREA valoare la sfârșitul corpului
    }
    cout << s;
    return 0;
}
```



Suma din mers

Urmărirea variabilelor pentru intrarea `5 3 8 0`:

moment	x	`x != 0`?	s
după prima citire	5	da	0
după iterația 1	3	da	5
după iterația 2	8	da	8
după iterația 3	0	**nu** → ies	16

Intrare / ieșire:

Intrare	Ieșire	Explicație
`5 3 8 0`	`16`	5 + 3 + 8
`0`	`0`	corpul nu se execută deloc
`7 0`	`7`	o singură iterație

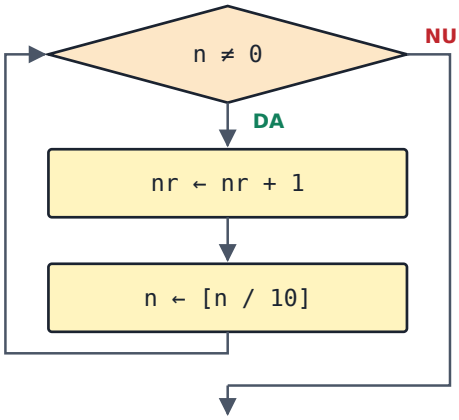
DE CE AȘA?

De ce citim de două ori? Trebuie să știm valoarea lui `x` **înainte** de a testa condiția, deci prima citire e în afara buclei. Apoi, la sfârșitul fiecărei iterații, citim următoarea valoare ca să fie testată la reluare. Dacă am citi la început de corp și apoi am testa, am prelucra și valoarea 0 (inofensiv la sumă, dar greșit la numărări și la produse).

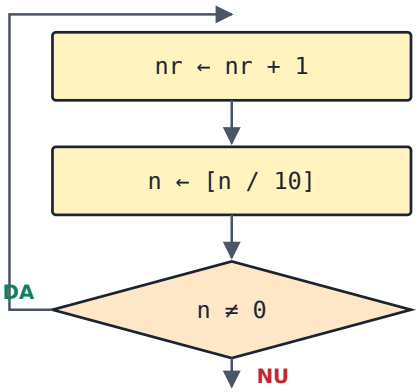
3. `do-while` — testul după

```
do
{
    // corpul se execută cel puțin o dată
} while (conditie);       // atenție la ; de la final
```

Exemplu clasic: numărul de cifre. Funcționează și pentru `n = 0`, care are **o** cifră.



while: pentru n = 0 → 0 cifre (greșit)



do-while: pentru n = 0 → 1 cifră

while versus do-while

```cpp
#include <iostream>
using namespace std;

int main()
{
    int n, nr = 0;
    cin >> n;
    do
    {
        nr++;
        n = n / 10;
    } while (n != 0);
    cout << nr;
    return 0;
}
```

iterația	n la început	nr după	n după
1	2048	1	204
2	204	2	20
3	20	3	2
4	2	4	0 → stop

DE CE AȘA?

De ce `do-while` aici? Orice număr, inclusiv 0, are cel puțin o cifră. Cu `while (n != 0)` pentru `n = 0` bucla nu rulează și am afișa 0 cifre. Folosim `do-while` exact când acțiunea trebuie făcută **măcar o dată** (alt exemplu: „citește până când utilizatorul introduce un număr valid").

4. `repetă … până când` din pseudocod

La Bac, pseudocodul folosește `repetă … până când`, care se **oprește** când condiția devine adevărată. În C++ nu există `until`, deci **negăm** condiția:

```
┌repetă                          do
│   s ← s + n % 10               {   s = s + n % 10;
│   n ← [n / 10]                      n = n / 10;
└până când n = 0                 } while (n != 0);   // NOT (n = 0)
```

ATENȚIE — GREȘELI FRECVENTE

`repetă … până când C` se oprește când **C e adevărată**; `do … while (C)` continuă cât **C e adevărată**. Condițiile sunt opuse: asta e cea mai frecventă greșeală la transcriere.

LA BAC

Exercițiu foarte frecvent la Subiectul al II-lea: „Scrieți un algoritm pseudocod echivalent cu cel dat, înlocuind structura *cât timp* cu o structură repetitivă de alt tip." Pași: 1. `cât timp C execută B` devine `dacă C atunci repetă B până când not C`; 2. Condiția se neagă (De Morgan dacă e compusă); 3. `dacă`-ul din față e necesar pentru că `repetă` execută corpul cel puțin o dată.

5. Bucle infinite — cum le recunoști

```cpp
int i = 1;
while (i <= 10)
    cout << i << ' ';     // lipsește i++ → afișează 1 la nesfârșit
```

```cpp
int n = 5;
while (n != 0)
    n = n - 2;            // 5, 3, 1, -1, -3... nu ajunge niciodată la 0
```

Pe pbinfo, o buclă infinită apare ca **„Depășire limită de timp"**. Verifică:

- se modifică variabila din condiție?
- se apropie de valoarea de oprire? Poate „sări" peste ea?

DE CE AȘA?

De ce `n != 0` e riscant, iar `n > 0` e mai sigur? `n != 0` cere ca valoarea să **aterizeze exact** pe 0. Cu `n > 0` e suficient să treacă dedesubt. Când nu ești sigur că pasul „nimerește" valoarea de oprire, folosește `<`, `>`.

6. Tipare de prelucrare „din mers"

Nu memorăm toate numerele, le prelucrăm pe rând, pe măsură ce le citim.

Ce calculăm	Inițializare	În buclă
sumă	`s = 0`	`s += x;`
produs	`p = 1`	`p *= x;`
câte au proprietatea P	`cnt = 0`	`if (P) cnt++;`
ultimul cu proprietatea P	`ult = -1`	`if (P) ult = x;`

Ce calculăm	Inițializare	În buclă
primul cu proprietatea P	`prim = -1`	`if (P && prim == -1) prim = x;`
există vreunul cu P?	`gasit = false`	`if (P) gasit = true;`
toate au P?	`toate = true`	`if (!P) toate = false;`

DE CE AȘA?

De ce suma pornește de la 0 și produsul de la 1? Valoarea inițială trebuie să fie **elementul neutru**: 0 nu schimbă o sumă, 1 nu schimbă un produs. Un produs care pornește de la 0 ar rămâne 0.

Exemplu: oprire la două valori consecutive egale (`#459 suma1`)

Trebuie să ținem minte valoarea **anterioară**:

```
#include <iostream>
using namespace std;

int main()
{
    int ant, x, s;
    cin >> ant >> x;
    s = ant + x;
    while (x != ant)
    {
        ant = x;          // valoarea curentă devine „anterioară”
        cin >> x;         // citim una nouă
        s += x;
    }
    cout << s;
    return 0;
}
```

citit	ant	x	`x != ant` ?	s
1 2	1	2	da	3
+3	2	3	da	6
+3	3	3	**nu** → stop	9

(Verifică pe enunț dacă trebuie adunate și cele două valori egale: fiecare problemă precizează.)

DE CE AȘA?

De ce `ant = x;` înainte de `cin >> x;` ? Dacă am citi mai întâi, am pierde vechea valoare. Mereu: salvezi vechea valoare, apoi citești noua.

7. Greșeli frecvente

Greșeala	Efect	Corect
`while (x != 0);`	`;` face corpul gol → buclă infinită	fără `;`
lipsește citirea de la final	bucla repetă aceeași valoare	`cin >> x;` la sfârșit
`do { } while (c)` fără `;`	eroare de compilare	`while (c);`
acumulator neinițializat	rezultat aleator	`s = 0`
`repetă…până când C` copiat ca `while (C)`	condiția e inversată	`while (!C)`

8. La BAC

> **LA BAC**
>
> Aproape sigur ai o grilă în care cauți **câte iterații** are o buclă sau **ce valoare** are o variabilă la ieșire. Fă un tabel cu coloanele variabilelor și completează rând cu rând (ca în lecție). Nu face în minte mai mult de două iterații.

Exemplu: `n = 2048`, numărăm zerourile.

n	`n % 10`	zero?	k
2048	8	nu	0
204	4	nu	0
20	0	**da**	1
2	2	nu	1
0		stop	1

9. Exerciții tip Bac (pe hârtie)

Răspunsurile sunt la finalul lecției, după probleme.

1. Ce afișează pentru `n = 2048`? Dar pentru `n = 1000`?

```
citește n
k ← 0
┌cât timp n > 0 execută
│   ┌dacă n % 10 = 0 atunci k ← k + 1
│   └■
│   n ← [n / 10]
└■
scrie k
```

2. Dați un exemplu de valoare pentru `n`, cu exact 4 cifre, pentru care se afișează 3.

3. Scrieți un algoritm echivalent folosind `repetă … până când`.

4. Ce afișează?

```cpp
int i = 1, s = 0;
while (i <= 5) { s += i; i += 2; }
cout << s << ' ' << i;
```

5. De câte ori se execută corpul buclei? Ce valoare are `i` după ea?

```cpp
int i = 10;
while (i > 0) i -= 3;
```

6. Care dintre secvențe sunt bucle **infinite**?

a) `i = 1; while (i <= 10) i--;`

b) `n = 7; while (n != 0) n -= 2;`

c) `i = 1; while (i <= 10) i *= 2;`

d) `x = 100; while (x > 0) x /= 2;`

7. Ce afișează `cout << i;` după secvența următoare?

```cpp
int s = 0, i = 1;
while (s < 20) { s += i; i++; }
```

8. Scrie bucla care citește numere până la 0 și afișează **câte numere pare** au fost citite.

9. Traduceți în C++: `repetă citește x; s ← s + x până când s ≥ 100`.

10. Mai departe (pentru cine vrea mai mult)

Conjectura lui Collatz. Pornește de la un număr `n`. Dacă e par, îl împarți la 2; dacă e impar, îl faci `3n + 1`. Repetă. Se pare că ajungi mereu la 1, dar **nimeni nu a demonstrat** asta! Verifică pe calculator:

```cpp
int n = 27, pasi = 0;
while (n != 1)
{
    if (n % 2 == 0) n = n / 2;
    else n = 3 * n + 1;
    pasi++;
}
cout << pasi;   // 111
```

Pentru `n = 27` numărul crește până la 9232 înainte să coboare. Aici condiția `n != 1` nu se poate înlocui ușor cu `<`: nu știi dacă bucla se termină, iar acest lucru e o problemă matematică deschisă.

Invariantul buclei. Cum demonstrezi că o buclă e corectă? Găsești o afirmație adevărată **înainte** de buclă, păstrată de fiecare iterație, și care la ieșire dă răspunsul. La suma de mai sus: „`s` = suma valorilor citite până acum". Este ideea din spatele demonstrațiilor de corectitudine din informatica serioasă.

Mai puțin cod, mai puține bug-uri. `while (cin >> x)` citește până la sfârșitul datelor (fără valoare-santinelă 0). Este varianta folosită la probleme de concurs; încearcă-o: `while (cin >> x) s += x;`.

Rezumat

- `while` testează înainte (poate executa de 0 ori); `do-while` testează după (cel puțin o dată).
- Citire până la 0: citește înainte, prelucrează, citește la final.
- `repetă … până când C` ≡ `do { } while (!C);`
- Acumulatori: sumă începe de la 0, produs de la 1, contor de la 0.
- Orice buclă trebuie să se apropie de condiția de oprire; urmărește variabilele într-un tabel.

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] `#51` **suma** — ușoară
  Citire până la 0, suma tuturor valorilor.
  https://www.pbinfo.ro/probleme/51/suma
- [ ] `#52` **sumapare1** — ușoară
  La fel, dar adunăm doar numerele pare.
  https://www.pbinfo.ro/probleme/52/sumapare1
- [ ] `#459` **suma1** — ușoară
  Oprire la două valori consecutive egale: trebuie să ții minte valoarea anterioară.
  https://www.pbinfo.ro/probleme/459/suma1

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] `#53` **cateimpare** — ușoară
  https://www.pbinfo.ro/probleme/53/cateimpare
- [ ] `#1261` **catepare** — ușoară
  https://www.pbinfo.ro/probleme/1261/catepare
- [ ] `#4278` **sumcnt1** — ușoară
  https://www.pbinfo.ro/probleme/4278/sumcnt1
- [ ] `#4921` **suma2cifre** — ușoară
  https://www.pbinfo.ro/probleme/4921/suma2cifre
- [ ] `#4560` **suma3cifre** — ușoară
  https://www.pbinfo.ro/probleme/4560/suma3cifre

Grile din subiectele de BAC (pbinfo)

Exerciții de tip grilă din variantele de Bacalaureat, rezolvate online pe pbinfo, cu verificare automată a răspunsului.

- [ ] #22 **Exercițiul #22**
  valoarea inițială a lui x ca bucla să se termine cu un rezultat dat (Bac 2009)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=22
- [ ] #40 **Exercițiul #40**
  de câte ori se execută afișarea (Bac 2009)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=40
- [ ] #43 **Exercițiul #43**
  ce valoare inițială trebuie să aibă x (Bac 2009)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=43
- [ ] #45 **Exercițiul #45**
  numărul de atribuiri efectuate (Bac 2009)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=45
- [ ] #25 **Exercițiul #25**
  câte atribuiri se execută în total (Bac 2009)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=25
- [ ] #798 **Exercițiul #798**
  pseudocod cu a%b în buclă (Bac 2017)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=798
- [ ] #806 **Exercițiul #806**
  pseudocod cu a%b în buclă (Bac 2017)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=806
- [ ] #1208 **Exercițiul #1208**
  pseudocod cu a%b în buclă (Bac 2021)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1208
- [ ] #1355 **Exercițiul #1355**
  secvență do-while (Bac 2022)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1355
- [ ] #1414 **Exercițiul #1414**
  n=19; while(n!=0): ce afișează (Bac 2023)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1414

Citește și exersează în plus

- Structuri repetitive https://www.pbinfo.ro/articole/71/structuri-repetitive
- Instrucțiunea while https://www.pbinfo.ro/articole/26009/instructiunea-while
- Instrucțiunea do...while https://www.pbinfo.ro/articole/26010/instructiunea-do-while
- Python Tutor pentru C++ — rulează un while pas cu pas și urmărește variabilele
  https://pythontutor.com/cpp.html

- cppreference — while / do-while — referința oficială, în engleză
  https://en.cppreference.com/w/cpp/language/while
- VisuAlgo — animații pentru algoritmi (mai folositor din lecțiile următoare) https://visualgo.net/en

Răspunsuri

Exercițiul de BAC (secțiunea 8)

Pentru `2048` se afișează `1`: un singur zero, cifra zecilor.

Exersează pe hârtie

1. Pentru `2048`: `1`. Pentru `1000`: `3` (trei zerouri; la ultima iterație n = 1 și 1 % 10 = 1).

2. De exemplu `1000`, `2000`, `9000` (trei zerouri la final). Numerele ca `1001` dau 2 și nu merg.

3.

```
citește n
k ← 0
┌dacă n > 0 atunci
│  ┌repetă
│  │   ┌dacă n % 10 = 0 atunci k ← k + 1
│  │   └■
│  │   n ← [n / 10]
│  └până când n ≤ 0
└■
scrie k
```

`dacă`-ul de la început este necesar: `repetă` execută corpul măcar o dată, iar `cât timp` poate să nu-l execute.

4. `9 7`. i ia valorile 1, 3, 5 (s = 1 + 3 + 5 = 9); după 5 devine 7 și condiția e falsă.

5. 4 execuții (i = 10, 7, 4, 1). După ele, `i = -2`.

6. a) și b). La a) `i` scade și rămâne mereu ≤ 10. La b) 7, 5, 3, 1, −1, ... nu atinge niciodată 0. La c) `i` merge 1, 2, 4, 8, 16 → se oprește. La d) x ajunge 100, 50, 25, 12, 6, 3, 1, 0 → se oprește.

7. `7`. s = 1, 3, 6, 10, 15, 21 (după i = 1, 2, 3, 4, 5, 6); al șaselea pas aduce s la 21 și i la 7.

8.

```cpp
int x, cnt = 0;
cin >> x;
while (x != 0)
{
    if (x % 2 == 0) cnt++;
    cin >> x;
}
cout << cnt;
```

9.

```cpp
int x, s = 0;
do
{
    cin >> x;
    s = s + x;
} while (s < 100);
```

`s ≥ 100` neagat este `s < 100`.
