PROGRAMARE ȘI STRUCTURI DE DATE

CURS 1

Lect. dr. Oneț - Marian Zsuzsanna

Facultatea de Matematică și Informatică UBB
în colaborare cu NTT Data

Cuprins

1. Organizarea cursului
2. Introducere în programare

Date de contact

- Adresă de e-mail: *marianzsu@cs.ubbcluj.ro*
  - Dacă aveți orice întrebări sau probleme legate de această disciplină, nu ezitați să mă contactați!
- Pagina personală: *www.ubbcluj.ro/~marianzsu/PStructDate.html*
  - Cursuri, seminarii, teme de laborator
  - Alte materiale
- Consultații:
  - De stabilit ...

Obiectivele cursului

- Introducere în modul de calcul a complexității algoritmilor
- Cunoașterea Tipurilor Abstracte de Date (TAD) Containere și a operațiilor specifice
- Cunoașterea structurilor de date cu care putem implementa aceste containere, împreună cu avantaje și dezavantaje

Evaluare

- Nota finală este alcătuită din 3 componente:
  - Examen scris (60%)
  - Notă proiect (30%)
  - Notă laborator (10%)
- Pentru a promova, trebuie să aveți minim nota 5 (fără rotunjire) la:
  - Examen scris
  - Proiect
  - Nota finală
- Pentru activitatea de seminar veți primi puncte bonus, care vor fi adunate la nota finală (maximum 1 punct in total)

Introducere în programare I

- Pentru a rezolva o problemă cu ajutorul calculatorului avem nevoie de un *algoritm*.
- Un algoritm este o secvență de instrucțiuni folosită pentru a rezolva o problemă.
- Pot exista mai mulți algoritmi corecți pentru a rezolva o problemă.
- Un program este implementarea unui algoritm într-un limbaj de programare.

Introducere în programare II

- Pași pentru rezolvarea unei probleme complexe:
  - Formularea problemei și a cerințelor
  - Specificarea problemei
  - Planificarea/alegerea algoritmilor
  - Implementarea algoritmilor într-un limbaj de programare
  - Testarea programului
  - Scrierea documentației
  - Utilizarea și mentenanța programului

Introducere în programare III

- *Un algoritm este o secvență de* **instrucțiuni** *folosită pentru a rezolva o problemă*.
  - Există instrucțiuni specifice pentru calculatoare și trebuie să construim algoritmi folosind aceste instrucțiuni

Pseudocod

- Pseudocodul este un limbaj de descriere al algoritmilor, destinat cititorului uman, nu calculatorului
- Avantajele pseudocodului:
  - Este mai informal decât un limbaj de programare
  - Este mai ușor de înțeles, pentru că nu conține elemente specifice unei limbaj de programare
  - Un algoritm descris în pseudocod poate fi transcris ușor într-un limbaj de programare
- Dezavantajul presudocodului:
  - Pseudocodul nu poate fi executat

Un algoritm

- Un algoritm este o secvență de instrucțiuni folosită pentru a rezolva o problemă
- În general un algoritm are *date de intrare* - date/informații care sunt cunoscute când execuția algoritmului începe
- În general un algorithm are *date de ieșire* - date care reprezintă rezultatul algoritmului
- Transformarea datelor de intrare în date de ieșire se face printr-o secvență de instrucțiuni specifice calculatoarelor.

Variabile I

- Ce este o variabilă?

Variabile I

- Ce este o variabilă?
- Variabilele sunt folosite pentru a stoca datele de intrare, datele de ieșire și valori intermediare necesare pentru a rezolva problema.
- O variabilă are **un nume**
  - Acest nume este folosit pentru a se referi la variabila respectivă.
  - Nu putem avea 2 variabile cu aceeași nume într-un algoritm.
  - Numele trebuie să fie ales în așa fel încât să sugereze pentru ce vom folosi variabila.

Variabile II

- O variabilă are **o semnificație**
  - Fiecare variabilă reprezintă date sau informații din algoritmul nostru.
  - Este important ca să nu folosim aceeași variabilă să reprezinte date cu semnificații diferite.
- O variabilă are **un tip** (**un domeniu**)
  - Tipul variabilei arată ce valori pot fi stocate în această variabilă și ce operații pot fi efectuate folosind variabila.
  - Există tipuri predefinite în fiecare limbaj de programare, dar în general pot fi definite și tipuri noi.

Variabile III

- O variabilă are **o valoare**
  - Această valoare trebuie să corespundă tipului/domeniului variabilei.
  - Dacă o variabilă nu are valoare, se spune că este *neinițializată*.
  - Valoarea unei variabile poate fi modificată pe parcursul execuției algoritmului.
- O variabilă are **o adresă de memorie**
  - Este zona de memorie unde calculatorul stochează valoarea variabilei.
  - În anumite limbaje de programare avem acces la adresa unei variabile, în altele nu.

Constante

- Constantele sunt similare cu variabile, cu diferența că nu putem modifica valoarea unei constante.
- Sunt folosite pentru a stoca valori care nu se modifică (sau se modifică foarte rar), de ex. valoarea lui PI, câte zile sunt într-o săptămână, rata TVA-ului, etc.
- Avantajele folosirii constantelor:
  - Codul e mai lizibil, mai ușor de înțeles
  - Dacă totuși trebuie schimbată valoarea, schimbarea se face într-un singur loc

Tipuri de date și operații I

- Majoritatea limbajelor de programare au anumite tipuri de date predefinite împreună cu operații care pot fi efectuate pe valorile tipului.
- Ce tipuri de date cunoașteți?

Tipuri de date și operații I

- Majoritatea limbajelor de programare au anumite tipuri de date predefinite împreună cu operații care pot fi efectuate pe valorile tipului.
- Ce tipuri de date cunoașteți?
- În pseudocod presupunem că avem următoarele tipuri de date:
  - Întreg
  - Real
  - Boolean
  - Caracter
  - String (șir de caractere)
- În multe limbaje de programare, programatorul poate să definească alte tipuri de date prin combinarea tipurilor existente (de ex. tip număr rațional, definit prin 2 numere întregi)

Tipuri de date și operații II

Tipul Întreg

- Reprezintă valori întregi (pozitive și negative), de ex. 0, 1,10, 512, -6124, ...
- Operații posibile:
  - adunare (+), scădere (-), înmulțire (*)
  - împărțire întreagă (/) (de ex. $13 / 5 = 2$)
  - comparații: $=$, $!=$, $<$, $>$, $<=$, $>=$
  - mod (sau %): restul împărțirii (de ex. $13 \bmod 5 = 3$)

Tipuri de date și operații III

Tipul Real

- Reprezintă valori reale (pozitive și negative), de ex. 0.4, -0.21, 5.0, 0.00154, -65.02, ...
- Operații posibile:
  - adunare (+), scădere (-), înmulțire (*)
  - împărțire (/) (de ex. 13.0 / 5.0 = 2.6)
  - comparații: =, !=, <, >, <=, >=

Tipuri de date și operații IV

Tipul Boolean

- Variabile de tip Boolean pot avea doar 2 valori posibile: *Adevărat* sau *Fals*
- În anumite limbaje de programare valoarea 0 se consideră *Fals* și orice valoare diferită de 0 se consideră *Adevărat*
- Operații posibile:
  - - negație - **NOT**:

a	**not a**
Adevărat	Fals
Fals	Adevărat

- De exemplu: *Pentru a intra în examen trebuia să* **nu** *aveți mai mult de 2 absențe.*

Tipuri de date și operații V

- conjuncție - **ȘI**:

a	b	a și b
Adevărat	Adevărat	Adevărat
Adevărat	Fals	Fals
Fals	Adevărat	Fals
Fals	Fals	Fals

- De exemplu: *Pentru a promova această disciplină trebuie să aveți minim nota 5 la proiect* **și** *să aveți minim nota 5 la examenul scris.*

Tipuri de date și operații VI

- disjuncție - **SAU**:

a	b	a sau b
Adevărat	Adevărat	Adevărat
Adevărat	Fals	Adevărat
Fals	Adevărat	Adevărat
Fals	Fals	Fals

- De exemplu: *O persoană se poate pensiona dacă are 60 de ani* **sau** *a lucrat 40 de ani.*

Tipuri de date și operații VII

- disjuncție exclusivă - **XOR**:

a	**b**	**a xor b**
Adevărat	Adevărat	Fals
Adevărat	Fals	Adevărat
Fals	Adevărat	Adevărat
Fals	Fals	Fals

- În limba română nu există cuvânt separat pentru disjuncție exclusivă, tot *sau* este folosit.

Tipuri de date și operații VIII

Tipul Caractere

- Reprezintă un singur caracter, care poate fi o cifră, o literă (mică sau mare), un semn special (%, *, etc.) .
- În general fiecare caracter are un număr întreg asociat și este reținut în memorie sub forma acestui număr

Tipuri de date și operații IX

Char	Ctrl	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
NUL	^@	0	00	space	32	20	@	64	40	`	96	60
SOH	^A	1	01	!	33	21	A	65	41	a	97	61
STX	^B	2	02	"	34	22	B	66	42	b	98	62
ETX	^C	3	03	#	35	23	C	67	43	c	99	63
EOT	^D	4	04	$	36	24	D	68	44	d	100	64
ENQ	^E	5	05	%	37	25	E	69	45	e	101	65
ACK	^F	6	06	&	38	26	F	70	46	f	102	66
BEL	^G	7	07	'	39	27	G	71	47	g	103	67
BS	^H	8	08	(	40	28	H	72	48	h	104	68
HT	^I	9	09	)	41	29	I	73	49	i	105	69
LF	^J	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	^K	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	^L	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	^M	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	^N	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	^O	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	^P	16	10	0	48	30	P	80	50	p	112	70
DC1	^Q	17	11	1	49	31	Q	81	51	q	113	71
DC2	^R	18	12	2	50	32	R	82	52	r	114	72
DC3	^S	19	13	3	51	33	S	83	53	s	115	73
DC4	^T	20	14	4	52	34	T	84	54	t	116	74
NAK	^U	21	15	5	53	35	U	85	55	u	117	75
SYN	^V	22	16	6	54	36	V	86	56	v	118	76
ETB	^W	23	17	7	55	37	W	87	57	w	119	77
CAN	^X	24	18	8	56	38	X	88	58	x	120	78
EM	^Y	25	19	9	57	39	Y	89	59	y	121	79
SUB	^Z	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	^[	27	1B	;	59	3B	[	91	5B	{	123	7B
FS	^\	28	1C	<	60	3C	\	92	5C	|	124	7C
GS	^]	29	1D	=	61	3D	]	93	5D	}	125	7D
RS	^^	30	1E	>	62	3E	^	94	5E	~	126	7E
US	^_	31	1F	?	63	3F	_	95	5F	delete	127	7F

Tipuri de date și operații X

- Operații posibile:
  - Comparație a 2 caractere
  - Adunarea unui număr la un caracter: 'A' + 7 = 'H' (65 + 7 = 72) sau 'A' + 35 = 'd' (65 + 35 = 100)
  - Scăderea unui număr dintr-un caracter: 'a' - 32 = 'A'(97 - 32 = 65) sau '7' - 4 = '3' (55 - 4 = 51)
  - Verificarea dacă un caracter c este o cifră: '0' $\leq c \leq$ '9'
  - Verificarea dacă un caracter c este literă mică: 'a' $\leq c \leq$ 'z'
  - Transformarea unui caracter c care reprezintă o cifră în cifra respectivă: c - '0'
  - Transformarea unui caracter c care reprezintă o literă mică în numărul de ordine a literei: c - 'a' (va returna 0 pentru 'a', 1 pentru 'b', 2 pentru 'c', etc.)

Tipuri de date și operații XI

Tipul String (șir de caractere)

- Reprezintă o secvență de caractere folosită pentru a reține date de tip text. De exemplu: “Rezultatul”, “abc”, “Ianuarie”, etc.
- Operații posibile:
  - Determinarea numărului de caractere (lungimea șirului)
  - Accesarea caracterului de pe o poziție, accesarea substringului dintre 2 poziții
  - Concatenarea a 2 stringuri
  - Comparație de stringuri (folosind ordinea alfabetică)
  - etc.

Tipuri de date și operații XII

- În pseudocod pentru a defini o variabilă de un anumit tip vom folosi notația: **nume_variabila: tip**
- De exemplu:
  - nume: string
  - varsta: întreg
  - greutate: real
  - initiala_parinte: caracter
  - permis: boolean //presupunem că verificăm dacă are permis de conducere sau nu
- Când vrem să definim mai multe variabile de un anumit tip, vom scrie:
  - nume, prenume, oras: string

Afișare și citire

- Sunt metode prin care utilizatorul poate să comunice cu programul
- Prin citire introducem datele cu care vrem să lucreze programul. Operația de citire necesită furnizarea variabilei în care vrem să fie stocată valoarea citită.
- Prin afișare putem vedea rezultatele programului pe ecran. Operația de afișare necesită furnizarea valorii de afișat, sub forma de variabilă sau sub forma de valoare.
- În pseudocod pentru citire folosim instrucțiunea **citește nume_variabilă** iar pentru afișare folosim **scrie nume_variabilă** sau **scrie valoare**.

Hello World

Exemplu 1 - HelloWorld

algoritm HelloWorld **este**
    scrie “Hello World”
sf_algoritm

- Să rescriem algoritmul, astfel încât să salutăm utilizatorul.

Exemplu 2 - HelloUtilizator

algoritm HelloUtilizator **este**
    nume: string
    scrie "Cum te numești?"
    citește nume
    scrie "Hello " + nume
// semnul + reprezintă concatenare de stringuri
sf_algoritm

- // marchează începutul unui **comentariu**
- Comentariile sunt texte care sunt ignorate de calculator când codul este compilat sau rulat, dar sunt importante pentru programatori: ajută la înțelegerea codului.
- Folosirea comentariilor este importantă!
- E bine ca înainte de o instrucțiune de citire să afișăm un mesaj care să sugereze ce vrem să citim.

Atribuire I

- Se folosește pentru a seta valoarea unei variabile la o anumită valoare.
- În momentul în care se declară o variabilă este bine ca aceasta să fie inițializată (adică să primească o valoare).
- În pseudocod folosim notația **variabila** = **valoare**. De exemplu:
  - nume = “Maria”
  - lungime = 10
  - PI = 3.1415
  - suma = 10+3

Atribuire II

- Valoarea atribuită poate să fie rezultatul unei expresii și poate să depindă de valoarea altor variabile. De ex (*presupunem că n, a și b sunt variabile existente*):
  - nume = "Maria" + " Alina"
  - lungime = 2 * n
  - suma = a + b
- Prima dată se evaluează partea dreaptă a expresiei, iar rezultatul evaluării este atribuit variabilei din partea stângă. De aceea, putem scrie expresii de genul: $v = v + 1$
  - Prima dată se evaluează $v + 1$, după care rezultatul i se atribuie lui v.

Atribuire III

- Pentru a verifica dacă valoarea unei variabile este egală cu o anumită valoare, se folosește notația **variabilă == valoare**. De exemplu:
  - nume == “Maria”
  - lungime == 10
  - suma == 13
  - a == suma (presupunem că există variabila *suma*)
- Pentru a verifica dacă valoarea unei varibile este diferită de o anumită valoare, se folosește notația **variabila != valoare**. De exemplu:
  - nume != “Maria”
  - suma != 13
  - suma != a (presupunem că există variabila *a*)

Arie și perimetru dreptunghi

- Să se calculeze aria și perimetrul unui dreptunghi cu laturi a și b.

Arie și perimetru dreptunghi

- Să se calculeze aria și perimetrul unui dreptunghi cu laturi a și b.
- Formula pentru a calcula aria unui dreptunghi cu laturi a și b este: $a * b$
- Formula pentru a calcula perimetrul unui dreptunghi cu laturi a și b este: $2 * (a + b)$

Exemplu 3 - AriePerimetruDreptunghi

algoritm AriePerimetruDreptunghi **este**
    a, b, arie, perimetru: real
    scrie "Cât este lungimea dreptunghiului?"
    citește a
    scrie "Cât este lățimea dreptunghiului?"
    citește b
    arie = a * b
    perimetru = 2 * (a + b)
    scrie "Aria este: " + arie + " perimetrul este: " + perimetru
sf_algoritm

Instrucțiuni condiționale

- Să se considere următoarea problemă: *Să se citească de la tastatură un număr pozitiv și să se afișeze un mesaj din care să reiasă dacă numărul este par sau impar.*
- Cum verificăm dacă un număr este par sau nu?

Instrucțiuni condiționale

- Să se considere următoarea problemă: *Să se citească de la tastatură un număr pozitiv și să se afișeze un mesaj din care să reiasă dacă numărul este par sau impar.*
- Cum verificăm dacă un număr este par sau nu?
- Pentru a verifica dacă un număr este par sau impar se verifică restul împărțirii la 2 (operația *mod*). Dacă restul este 0, numărul este par, dacă restul este 1, numărul este impar.

Instrucțiuni condiționale II

- De foarte multe ori avem de efectuat operații diferite în funcție de anumite condiții. În aceste situații folosim instrucțiunea condițională:

dacă condiție **atunci**
    instrucțiune
altfel
    instrucțiune
sf_dacă

- La partea *instrucțiune* se poate scrie o singură instrucțiune, sau mai multe instrucțiuni.

Instrucțiuni condiționale III

- Partea *altfel* poate să lipsească, dacă nu avem nimic de executat pe ramura respectivă:

dacă condiție **atunci**
    instrucțiune
sf_dacă

Instrucțiuni condiționale III

- Partea *altfel* poate să lipsească, dacă nu avem nimic de executat pe ramura respectivă:

dacă condiție **atunci**
    instrucțiune
sf_dacă

- Pe partea *altfel* putem adăuga încă o condiție:

dacă condiție1 **atunci**
    instrucțiune
altfel dacă condiție2 **atunci**
    instrucțiune
altfel
    instrucțiune
sf_dacă

Număr par sau impar

Exemplu 4 - ParImpar

```
algoritm ParImpar este
    număr, rest: întreg
    scrie "Introduceți numărul:"
    citește număr
    rest = număr mod 2
    dacă rest == 0 atunci
        scrie "Numărul este par"
    altfel
        scrie "Numărul este impar"
    sf_dacă
sf_algoritm
```

An bisect

- Să considerăm următoarea problemă: *Să se citească de la tastatură un an și să se afișeze câte zile a avut anul respectiv.*

An bisect

- Să considerăm următoarea problemă: *Să se citească de la tastatură un an și să se afișeze câte zile a avut anul respectiv.*
- Numărul de zile într-un an este 366 dacă anul e bisect și 365 dacă anul nu e bisect.
- Când este un an bisect?

An bisect

- Să considerăm următoarea problemă: *Să se citească de la tastatură un an și să se afișeze câte zile a avut anul respectiv.*
- Numărul de zile într-un an este 366 dacă anul e bisect și 365 dacă anul nu e bisect.
- Când este un an bisect?
- Un an este bisect dacă este divizibil cu 4 și nu este divizibil cu 100, sau dacă este divizibil cu 400. De exemplu: 2000, 2004, 2016, 2104, etc. sunt ani bisecți, dar 2001, 2002, 1900, 2100, 2200, etc. nu sunt.

An bisect II

Exemplu 5 - AnBisect

algoritm AnBisect **este**
    an, rez: întreg
    scrie "Introduceți anul:"
    citește an
    rez = 365 *//presupunem că nu e an bisect*
    dacă an **mod** 400 == 0 SAU (an **mod** 4 == 0 ȘI an **mod** 100 !=0) **atunci**
        rez = 366
    sf_dacă
    scrie "Numărul de zile este " + rez
sf_algoritm

Instrucțiuni repetitive

- Să considerăm următoarea problemă: *Să se citească de la tastatură 10 numere și să se calculeze suma și media lor. De exemplu, dacă numerele citite sunt 7, 13, 62, 61, -32, 9, 85, 1, 56, 42, suma lor este 304, iar media lor este 30.4.*
- Deși știm sigur că este nevoie de 10 numere, a lua 10 variabile și a scrie instrucțiunea de citire de 10 ori nu pare o soluție foarte bună.
- Când avem de repetat aceleași instrucțiuni de mai multe ori, vom folosi instrucțiuni repetitive: ciclu **pentru** sau ciclu **câttimp**

Instrucțiunea **pentru** I

- Se folosește când știm de câte ori trebuie să repetăm anumite instrucțiuni
- în pseudocod instrucțiunea **pentru** arată în modul următor:

pentru variabilă = val_inceput, val_final, pas **execută**
    instrucțiune
sf_pentru

- *variabilă* se numește variabilă de ciclu. În momentul în care începe execuția ciclului, *variabilă* primește valoarea *val_inceput*.

Instrucțiunea **pentru** II

- La fiecare iterație, se verifică dacă *variabilă* are valoare egală sau mai mare decât *val_final*. Dacă da, se oprește ciclul. Dacă nu, se execută intrucțiunile din interiorul ciclului, iar valoarea lui *variabilă* se modifică cu valoarea *pasului* (practic *variabilă* = *variabilă* + *pas*).
- Dacă *pas* are o valoarea negativă, atunci execuția ciclului se oprește când *variabilă* are valoarea mai mică decât *val_final*.
- Când *variabilă* == *val_final* ciclul nu se mai execută.
- Dacă *val_început* este egală sau mai mare decât *val_final* (sau mai mic, dacă *pas* e negativ), ciclul nu se execută deloc.

Sumă și medie

Exemplu 6 - SumaSiMedie

algoritm SumaȘiMedie **este**
  suma, contor, număr: întreg
  medie: real
  suma = 0 // *inițializăm suma*
  medie = 0.0 // *și media*
  pentru contor = 1, 11, 1 **execută**
    scrie “Dați un număr: “
    citește număr
    suma = suma + număr
  sf_pentru
  medie = suma / 10.0
  scrie “Suma este: “ + suma
  scrie “Media este: “ + medie
sf_algoritm

Sumă și medie II

- Ce se întâmplă dacă nu 10 numere trebuie citite ci n? (Unde n este o valoare care trebuie citită la începutul algoritmului)

Exemplu 7 - SumaSiMedie2

algoritm SumaȘiMedie2 **este**
    suma, contor, număr, câte: întreg
    medie: real
    suma = 0 // *inițializăm suma*
    medie = 0.0 // *și media*
    scrie "Numărul de numere care vor fi citite: "
    citește câte
    pentru contor = 1, câte+1, 1 **execută**
        scrie "Dați un număr: "
        citește număr
        suma = suma + număr
    sf_pentru
    medie = suma / (câte * 1.0) //*forțez împărțirea cu număr real*
    scrie "Suma este: " + suma
    scrie "Media este: " + medie
sf_algoritm

Sumă și medie II

- Este corect codul nostru? Vom avea soluția corectă de fiecare dată? Pentru fiecare valoare a lui n?

Sumă și medie II

- Este corect codul nostru? Vom avea soluția corectă de fiecare dată? Pentru fiecare valoare a lui *n*?
- Avem probleme dacă utilizatorul introduce valoarea 0 pentru *n*. Împărțirea la 0 este o operație care cauzează eroare. De aceea, înainte de a face o împărțire, trebuie să verificăm să nu împărțim la 0.
- Dacă *n* este 0, atunci nu facem împărțirea, și *medie* rămâne 0.

Sumă și medie II

Exemplu 8 - SumaSiMedie2

algoritm SumaȘiMedie2 **este**
    suma, contor, număr, câte: întreg
    medie: real
    suma = 0 // *inițializăm suma*
    medie = 0.0 // *și media*
    scrie "Numărul de numere care vor fi citite: "
    citește câte
    pentru contor = 1, câte+1, 1 **execută**
        scrie "Dați un număr: "
        citește număr
        suma = suma + număr
    sf_pentru
    dacă câte != 0 **atunci**
        medie = suma / (câte * 1.0) //*forțez împărțirea cu număr real*
    sf_dacă
    scrie "Suma este: " + suma
    scrie "Media este: " + medie
sf_algoritm

Instrucțiunea **câttimp** I

- Ce se întâmplă dacă nu se știe de la început numărul de numere de adunat, ci se știe că trebuie să citim numere până când utilizatorul introduce numărul 0?
- Dacă nu știm de câte ori trebuie să repetăm instrucțiunile, nu putem folosi ciclu **pentru**.
- În situațiile în care anumite instrucțiuni trebuie repetate până când o condiție devine adevărată (sau falsă), se folosește ciclul **câttimp**.

Instrucțiunea **câttimp** II

câttimp condiție **execută**
    instrucțiune
sf_câttimp

- La intrare în ciclul **câttimp** se verifică condiția. Dacă este adevărată, se execută instrucțiunile din ciclu. După aceea, se reverifică condiția. Dacă este adevărată, se execută din nou instrucțiunile din ciclu. După care iar se verifică condiția și așa mai departe.
- În momentul în care condiția este falsă, nu se mai execută instrucțiunile din ciclu, execuția continuă cu instrucțiunile după **sf_câttimp**.

Instrucțiunea **câttimp** III

- Dacă condiția este falsă de la început, instrucțiunile din ciclul **câttimp** nu se execută deloc.
- Instrucțiunea (sau instrucțiunile) din ciclul **câttimp** trebuie să conțină elemente care să modifice condiția, altfel, dacă condiția nu devine falsă niciodată, vom avea un ciclu infinit!

Sumă și medie III

Exemplu 9 - SumaSiMedie3

algoritm SumaȘiMedie3 **este**
suma, contor, număr, câte: întreg
medie: real
suma = 0 // *inițializăm suma*
medie = 0.0 // *și media*
câte = 0 //*trebuie să număr câte numere sunt citite, pentru a calcula media*
scrie "Dați un număr: "
citește număr
câttimp număr != 0 **execută**
suma = suma + număr
câte = câte + 1 //*incrementez numărul de numere citite*
scrie "Dați un număr: " //*citesc o nouă valoare pentru număr*
citește număr
sf_câttimp
//*continuare pe pagina următoare...*

```
   dacă câte != 0 atunci
      medie = suma / (câte * 1.0)
   sf_dacă
   scrie "Suma este: " + suma
   scrie "Media este: " + medie
sf_algoritm
```

Sumă și medie III

- Bucata de cod care afișează mesajul "'Dați un număr" și care citește numărul apare de 2 ori în codul nostru. Dacă vrem să nu scriem de 2 ori aceste instrucțiuni, putem face citire doar în ciclul **câttimp**. În acest caz citirea se face la începutul ciclului (înainte de adăugare). Trebuie să fim atenți să inițializăm variabila *număr* în așa fel încât execuția să intre în *câttimp*.

Sumă și medie III

Exemplu 10 - SumaSiMedie3

algoritm SumaȘiMedie3 **este**
    suma, contor, număr, câte: întreg
    medie: real
    suma = 0 // *inițializăm suma*
    medie = 0.0 // *și media*
    număr = 1 //*putem inițializa variabila cu orice număr diferit de 0*
    câte = 0 //*trebuie să număr câte numere sunt citite, pentru a calcula media*
    câttimp număr != 0 **execută**
        scrie "Dați un număr: " //*citesc o nouă valoare pentru număr*
        citește număr
        suma = suma + număr
        câte = câte + 1 //*incrementez numărul de numere citite*
    sf_câttimp
    dacă câte != 0 **atunci**
        medie = suma / (câte * 1.0)
    sf_dacă
    scrie "Suma este: " + suma
    scrie "Media este: " + medie

Sumă și medie III

- Cât va fi rezultatul dacă introducem pe rând numerele 10, 51, 3, 0?

Sumă și medie III

- Cât va fi rezultatul dacă introducem pe rând numerele 10, 51, 3, 0?
- În loc să avem rezultatul 64 și 21.333, rezultatul afișat va fi 64 și 16. Motivul este faptul că momentan și valoarea 0 este numărată de variabila câte, și împărțirea se face la 4, nu la 3.
- Există două soluții:
  - să inițializăm variabila câte cu -1 astfel, la final va avea exact valoarea care ne trebuie
  - să facem împărțirea la câte - 1, și să verificăm dacă câte are valoarea 1, nu 0.

Sumă și medie III

Exemplu 11 - SumaSiMedie3 - soluția 1

algoritm SumaȘiMedie3 **este**
suma, contor, număr, câte: întreg
medie: real
suma = 0 // *inițializăm suma*
medie = 0.0 // *și media*
număr = 1 //*putem inițializa variabila cu orice număr diferit de 0*
câte = -1 //*trebuie să număr câte numere sunt citite, pentru a calcula*
//*media. Inițializăm cu -1, pentru a avea la final valoarea bună*
câttimp număr != 0 **execută**
scrie "Dați un număr: " //*citesc o nouă valoare pentru număr*
citește număr
suma = suma + număr
câte = câte + 1 //*incrementez numărul de numere citite*
sf_câttimp
dacă câte != 0 **atunci**
medie = suma / (câte * 1.0)
sf_dacă
//*continuare pe pagina următoare...*

scrie “Suma este: “ + suma
scrie “Media este: “ + medie
sf_algoritm

Sumă și medie III

Exemplu 11 - SumaSiMedie3 - soluția 2

algoritm SumaȘiMedie3 **este**
suma, contor, număr, câte: întreg
medie: real
suma = 0 // *inițializăm suma*
medie = 0.0 // *și media*
număr = 1 //*putem inițializa variabila cu orice număr diferit de 0*
câte = 0 //*trebuie să număr câte numere sunt citite, pentru a calcula media*
câttimp număr != 0 **execută**
scrie "Dați un număr: " //*citesc o nouă valoare pentru număr*
citește număr
suma = suma + număr
câte = câte + 1 //*incrementez numărul de numere citite*
sf_câttimp
dacă câte != 1 **atunci**
medie = suma / (câte * 1.0 - 1)
sf_dacă
scrie "Suma este: " + suma
scrie "Media este: " + medie

Cicluri **pentru** și **câttimp** I

- Orice ciclu **pentru** poate fi rescris ca un ciclu **câttimp**:
  - inițializăm o variabilă cu valoarea **val_început** înainte de ciclul **câttimp**
  - condiția din ciclul **câttimp** va verifica dacă valoarea variabilei e mai mică (mai mare) decât *val_final*
  - în interiorul ciclului **câttimp** modificăm valoarea variabilei cu valoarea *pasului*
- Deci, putem scrie soluția pentru *SumăȘiMedie2* - când știam că vom lucra cu *n* numere - folosind ciclul **câttimp**

Cicluri **pentru** și **câttimp** II

```
algoritm SumaȘiMedie2 este
    suma, contor, număr, câte: întreg
    medie: real
    suma = 0 // inițializăm suma
    medie = 0.0 // și media
    scrie "Numărul de numere care vor fi citite: "
    citește câte
    contor = 1
    câttimp contor < câte+1 execută
        scrie "Dați un număr: "
        citește număr
        suma = suma + număr
        contor = contor + 1
    sf_câttimp
    dacă câte != 0 atunci
        medie = suma / (câte * 1.0)
    sf_dacă
    scrie "Suma este: " + suma
    scrie "Media este: " + medie
sf_algoritm
```