PROGRAMARE ȘI STRUCTURI DE DATE

CURS 2

Lect. dr. Oneț-Marian Zsuzsanna

Facultatea de Matematică și Informatică UBB
în colaborare cu NTT Data

Cuprins

1. Tablouri
2. Vizibilitatea variabilelor

Tablouri I

- Să considerăm problema următoare: *Se citesc n numere pozitive de la tastatură. Să se determine câte dintre numerele citite sunt strict mai mici decât media numerelor.*

- De exemplu dacă citim următoarele 9 numere: 102, 58, 21, 634, 299, 11, 648, 24, 77, suma lor este 1874, media lor este 208.22, și sunt 6 numere care sunt mai mici decât media (102, 58, 21, 11, 24, 77).

Tablouri II

- Știm deja cum se calculează media a n numere. Diferența față de problemele anterioare este că aici prima dată trebuie să calculăm media, și abia după ce știm media putem verifica numerele. Dar pentru acest lucru trebuie să reținem toate numerele undeva.

- Când trebuie să reținem mai multe elemente de același tip, trebuie să folosim un **tablou**.

Tablouri III

- Un **tablou** este o colecție de elemente de același tip, care ocupă un spațiu continuu de memorie.
- Tabloul se mai numește șir, vector, array.
- Pentru a defini un tablou trebuie să specificăm ce tip vor avea elementele tabloului și câte elemente vrem să existe în tablou.

Tablouri IV

- În pseudocod un tablou se definește folosind notația:

nume_variabilă: tip[]
sau
nume_variabilă: tip[N]
de exemplu:
elemente: întreg[]
luni: întreg[12]

- În exemple și explicații, vom reprezenta tabloul folosind paranteze drepte și enumerând elementele. De exemplu: [102, 58, 21, 634, 299, 11, 648, 24, 77].

Tablouri V

- Accesarea elementelor dintr-un tablou se face pe baza pozițiilor. Primul element se găsește pe poziția 0, al doilea element pe poziția 1, etc.
- Ultimul element într-un tablou cu *n* elemente se găsește pe poziția *n-1*.
- Elementul de pe o poziție se accesează folosind paranteze drepte

Tablouri VI

scrie “primul element” + elemente[0]
scrie “al doilea element” + elemente[1]
scrie “al i-lea element” + elemente[i-1]
elemente[3] = 99
elemente[i] = elemente[i-1] + 1

- Înainte de a accesa un element de pe o poziție, trebuie să ne asigurăm că poziția este validă!

Mai mic ca medie

NumereMaiMiciCaMedie

```
algoritm NumereMaiMiciCaMedie este
    suma, nr, contor: întreg
    suma = 0
    scrie "Numărul de numere:"
    citește nr
    elemente: întreg[nr] //aici vom reține numerele
    pentru contor = 0, nr, 1 execută
        scrie "Dați un număr:"
        citește elemente[contor]
        suma = suma + elemente[contor]
    sf_pentru
    medie: real
    medie = 0
    dacă nr != 0 atunci
        medie = suma / (nr * 1.0)
    sf_dacă
    //continuare pe pagina următoare
```

```
    rezultat: întreg
    rezultat = 0
    //parcurgem din nou tabloul pentru a vedea care numere sunt mai mici
    // decât media
    pentru contor = 0, nr, 1 execută
        dacă elemente[contor] < medie atunci
            rezultat = rezultat + 1
        sf_dacă
    sf_pentru
    scrie "Sunt " + rezultat + " numere mai mici ca media"
sf_algoritm
```

String - un șir de caractere I

- Orice string poate fi considerat un tablou cu elemente de tip caracter (o singură literă).
- De exemplu, stringul “Maria”, poate fi considerat un tablou cu 5 elemente: [‘M’, ‘a’, ‘r’, ‘i’, ‘a’].
- Pentru a accesa un caracter de pe o poziție dintr-un string, folosim paranteze drepte (cum facem la tablouri).
- Pentru a accesa lungimea stringului (adică numărul de caractere din tablou), presupunem că avem funcția **lungime**.

String - un șir de caractere II

nume = “Maria” *//o variabilă de tip string, cu valoarea Maria*
nume[0] este ‘M’
nume[1] este ‘a’
...
lungime(nume) este 5
nume[**lungime**(nume)-1] este ‘a’ *//ultima literă din string*
nume[**lungime**(nume)] nu este o expresie validă - nu există caracter pe poziția respectivă

String - un șir de caractere III

- Să considerăm următoarea problemă: *CamelCase este o tehnică de a scrie cuvinte compuse sau fraze fără a folosi spațiu, scriind toate cuvintele împreună, dar începând fiecare cuvânt nou cu literă mare. CamelCase este folosit des de programatori, pentru a da nume de variabile, nume de funcții, etc. Un exemplu este chiar numele algoritmului scris mai înainte: NumereMaiMiciCaMedie. Citind de la tastatură un cuvânt scris cu CamelCase, să determinăm din câte cuvinte este alcătuit. Presupunem că prima literă este mică, deci primul cuvânt începe cu literă mică.*

String - un șir de caractere IV

- De exemplu:
  - *numeVariabilă* conține 2 cuvinte
  - *variabilă* conține 1 cuvânt
  - *numeLungSiComplicat* conține 4 cuvinte
  - *numeSiMaiLungSiMaiComplicat* conține 7 cuvinte
- Cum putem rezolva problema?

String - un șir de caractere V

- Din moment ce știm că fiecare cuvânt începe cu o literă mare, și că în fiecare cuvânt doar prima literă e mare, trebuie să numărăm câte litere mari sunt în cuvânt. Dar să nu uităm că primul cuvânt începe cu literă mică, deci rezultatul este cu 1 mai mult decât numărul de litere mari.
- Pentru a verifica dacă o literă este mare, pur și simplu îl comparăm cu prima și cu ultima literă mare din alfabet ('A' și 'Z'). Dacă este între aceste 2 litere, atunci e literă mare.

CamelCase

algoritm CamelCase **este**
  cuvânt: string
  nrCuvinte, poz: întreg
  scrie "Dați stringul:"
  citește cuvânt
  nrCuvinte = 1 *//sigur conține măcar un cuvânt*
  pentru poz = 0, **lungime**(cuvânt), 1 **execută**
    dacă cuvânt[poz] >= 'A' **ȘI** cuvânt[poz] <= 'Z' **atunci**
      nrCuvinte = nrCuvinte + 1
    sf_dacă
  sf_pentru
  scrie "Numărul de cuvinte: " + nrCuvinte
sf_algoritm

- Temă de gândire: *Cum ar trebui modificat codul, dacă nu știu dacă primul cuvânt va începe cu litera mică sau mare? Variabila cuvânt poate fi de exemplu NumeVariabilă sau numeVariabilă, în ambele cazuri rezultatul fiind 2.*

Tablouri multidimensionale I

- Tablourile de care am vorbit până acum erau unidimensionale, dar există și tablouri bidimensionale, tridimensionale, etc.
- Un tablou bidimensional, numit și matrice, conține 2 dimensiuni: linii și coloane
  - jocul Sudoku
  - tabla de șah
  - etc.
- Pentru a defini un tablou bidimensional este necesar să specificăm numărul de elemente pentru ambele dimensiuni (numărul de linii și numărul de coloane).

Tablouri multidimensionale II

- De exemplu, în pseudocod un tablou bidimensional se definește în modul următor:

```
matrice: întreg[N][N]
harta:întreg[10][30]
elemente: întreg[N][M]
```

- Un tablou bidimensional cu N linii și M coloane are în total N * M elemente.

Tablouri multidimensionale III

- Pentru accesarea unui element dintr-un tablou bidimensional va trebui să specificăm 2 valori: linia și coloana unde se găsește elementul. De exemplu:

Dacă matrice are N linii și M coloane:
matrice[0][0] este primul element din prima linie
matrice[0][M-1] este ultimul element din prima linie
matrice[5][9] este a 10-lea element din linia 6 (presupunem că există elementul)
matrice[N-1][0] este primul element din ultima linie
matrice[N-1][M-1] este ultimul element din ultima linie

Tablouri multidimensionale IV

- Să considerăm problema următoare: *Să se citească de la tastatură o matrice cu N linii și M coloane. Să se calculeze suma numerelor de pe linii pare și suma numerelor de pe coloane impare*.
- Pentru a lucra cu tablouri bidimensionale vom avea nevoie de 2 cicluri **pentru**, unul pentru indexul liniilor și unul pentru indexul coloanelor.

Tablouri multidimensionale V

- De exemplu:

	0	1	2	3	4	5	6	7
0	1	6	11	43	2	-5	17	21
1	-9	18	3	62	95	22	37	1
2	5	30	51	24	68	91	16	43
3	21	33	8	57	5	41	24	35
4	28	81	31	63	6	68	59	11
5	5	77	59	69	7	82	47	48
6	20	4	44	14	8	43	2	60

Tablouri multidimensionale VI

- Numere pe linii pare:

	0	1	2	3	4	5	6	7
0	1	6	11	43	2	-5	17	21
1	-9	18	3	62	95	22	37	1
2	5	30	51	24	68	91	16	43
3	21	33	8	57	5	41	24	35
4	28	81	31	63	6	68	59	11
5	5	77	59	69	7	82	47	48
6	20	4	44	14	8	43	2	60

Tablouri multidimensionale VII

- Numere pe coloane impare:

	0	1	2	3	4	5	6	7
0	1	6	11	43	2	-5	17	21
1	-9	18	3	62	95	22	37	1
2	5	30	51	24	68	91	16	43
3	21	33	8	57	5	41	24	35
4	28	81	31	63	6	68	59	11
5	5	77	59	69	7	82	47	48
6	20	4	44	14	8	43	2	60

SumaLiniiColoane

SumaLiniiColoane

algoritm SumaLiniiColoane **este**
  n, m, i, j: întreg
  scrie "Dimensiunile matricii:"
  citește n
  citește m
  matrice: întreg[n][m]
  //citirea elemenelor
  pentru i = 0, n, 1 **execută** *//i reprezintă linia curentă*
    pentru j = 0, m, 1 **execută** *//j reprezintă coloana curentă*
      scrie "Elementul de pe pozitia " + i + " " + j
      citește matrice[i][j]
    sf_pentru
  sf_pentru
//continuăm pe pagina următoare

SumaLiniiColoane II

```
    sumaL, sumaC: întreg
    sumaL = 0
    sumaC = 0
    //parcurgem matricea pentru a calcula sumele
    pentru i = 0, n, 1 execută
        pentru j = 0, m, 1 execută
            dacă i mod 2 == 0 atunci
                sumaL = sumaL + matrice[i][j]
            sf_dacă
            dacă j mod 2 == 1 atunci
                sumaC = sumaC + matrice[i][j]
            sf_dacă
        sf_pentru
    sf_pentru
    scrie "Suma liniilor pare: " + sumaL
    scrie "Suma coloanelor impare: " + sumaC
sf_algoritm
```

SumaLiniiColoane III

- Temă de gândire: *Ce se întâmplă dacă rescriu partea de comparație în modul următor:*

```
dacă i mod 2 == 0 atunci
    sumaL = sumaL + matrice[i][j]
altfel dacă j mod 2 == 1 atunci
    sumaC = sumaC + matrice[i][j]
sf_dacă
```

Tablouri multidimensionale

- Pentru a defini tablouri multidimensionale folosim următoarea instrucțiune:

```
tablouMultiD: întreg[D1][D2][D3]...[Dn]
```

- Iar pentru accesarea unui element, trebuie să folosim câte un indice pentru fiecare dimensiune:

```
tablouMultiD[0][0][0]...[0] = 0
```

Vizibilatatea variabilelor

- Vizibilitatea variabilelor descrie *viața* unei variabile, zona sa de disponibilitate (în care parte a codului poate fi folosită - este vizibilă) după ce a fost *definită*
- Considerăm că o variabilă este definită în momentul în care apare pentru prima oară în cod
  - Câteodată introducem o variabilă fără să-i atribuim o valoare (mai mult la tablouri)

Vizibilitatea variabilelor II

- O variabilă definită într-un algoritm (dar nu într-o instrucțiune *dacă*, *câttimp* sau *pentru*) este vizibilă în tot algoritmul după ce a fost definită.

Vizibilitatea variabilelor II

- O variabilă definită într-un algoritm (dar nu într-o instrucțiune *dacă*, *câttimp* sau *pentru*) este vizibilă în tot algoritmul după ce a fost definită.
- O variabilă definită într-un bloc (instrucțiune *dacă*, *câttimp* sau *pentru*) este vizibilă după ce a fost definită până la finalul blocului respectiv.

Vizibilitatea variabilelor II

- O variabilă definită într-un algoritm (dar nu într-o instrucțiune *dacă*, *câttimp* sau *pentru*) este vizibilă în tot algoritmul după ce a fost definită.
- O variabilă definită într-un bloc (instrucțiune *dacă*, *câttimp* sau *pentru*) este vizibilă după ce a fost definită până la finalul blocului respectiv.
- O variabilă definită în afara unui algoritm, numită *variabilă globală*, este vizibilă în toți algoritmii din fișierul respectiv. Este o practică greșită să folosim variabile globale!

Vizibilitatea variabilelor III

- Să considerăm problema următoare: *citiți un număr de la tastatură și afișați numărul de divizori ai numărului, urmat de lista divizorilor.*

Vizibilitatea variabilelor III

- Să considerăm problema următoare: *citiți un număr de la tastatură și afișați numărul de divizori ai numărului, urmat de lista divizorilor.*
  - Pentru a număra divizorii trebuie să verificăm pe rând fiecare număr între 1 și numărul respectiv, dacă este divizor.
  - Pentru a afișa divizorii, le vom reține într-un tablou, și le vom afișa după ce am afișat numărul de divizori.

Vizibilitatea variabilelor IV

```
1.algoritm Divizori este
2.    scrie "Dați un număr:"
3.    număr: întreg
4.    citește număr
5.    divizori: întreg[număr]  //tablou în care reținem divizorii
6.    nrDiv: întreg
7.    nrDiv = 0 // numărul de elemente din tabloul divizori
8.    divizorPosibil:întreg
9.    pentru divizorPosibil = 1, număr+1, 1 execută
10.       rest:întreg
11.       rest = număr mod divizorPosibil
12.       dacă rest == 0 atunci
13.           divizori[nrDiv] = divizorPosibil
14.           nrDiv = nrDiv + 1
15.       sf_dacă
16.   sf_pentru
17.   scrie "Numărul de divizori: " + nrDiv
```

Vizibilitatea variabilelor V

```
18.   scrie "Divizorii sunt: "
19.   div: întreg
20.   pentru div = 0, nrDiv, 1 execută
21.       scrie divizori[div]
22.   sf_pentru
23.sf_algoritm
```

Vizibilitatea variabilelor V

- Algoritmul *divizori* conține 6 variabile:
  - *număr* - definită pe rândul 3, vizibilă până la finalul algoritmului
  - *divizori* - definită pe rândul 5, vizibilă până la finalul algoritmului
  - *nrDiv* - definită pe rândul 6, vizibilă până la finalul algoritmului
  - *divizorPosibil* - definită pe rândul 8, vizibilă până la finalul algoritmului
  - *rest* - definită pe rândul 10, vizibilă doar în ciclul **pentru** (până rândul 16 inclusiv)
  - *div* - definită pe rândul 19, vizibilă până la finalul algoritmului

Vizibilitatea variabilelor VI

- Nu pot exista 2 variabile cu același nume vizibile în aceeași zonă a codului
- Dacă o variabilă nu mai este vizibilă, o altă variabilă cu același nume poate fi definită
- Încercarea de a folosi o variabilă care nu mai este vizibilă (precum și folosirea unei variabile care niciodată nu a fost definită), duce la o eroare

Exemple vizibilitatea variabilelor

- Ce va fi afișat dacă executăm următoarele bucăți de cod?

Exemplu 1

Exemplu 1

```
început: întreg
început = 6
sfârșit, i: întreg
sfârșit = 13
pentru i = început, sfârșit, 1, execută
   contor: întreg
    contor = 0
    dacă i mod 2 == 0 atunci
      contor = contor + 1
      scrie "Numere pare găsite până acum:" + contor
   sf_dacă
sf_pentru
scrie "Total numere pare: " + contor
```

Examplu 1

- Dacă lucrăm în limbajul Java (și rescriem bucata de cod în Java), vom avea o eroare la compilare, pentru că la final încercăm să afișăm variabila *contor* care nu mai este vizibilă.
- Dacă lucrăm în Python, codul va afișa mesajul *Numere pare găsite până acum: 1* de 4 ori, după care va da un mesaj de eroare la ultimul rând, pentru că variabila *contor* nu este vizibilă decât în ciclul *pentru*

Exemplu 2

Exemplu 2

nr, nrPași: întreg
nr = 129
nrPasi = 0
câttimp nr > 50 **execută**
    scrie "Numărul: " + nr
    rest: întreg
     rest = nr **mod** 2
     nr = (nr + rest) / 2
     nrPasi = nrPasi + 1
sf_câttimp
scrie "A fost nevoie de " + nrPasi + " pași"

Exemplu 2

Exemplu 2

```
nr, nrPași: întreg
nr = 129
nrPasi = 0
câttimp nr > 50 execută
    scrie "Numărul: " + nr
    rest: întreg
     rest = nr mod 2
     nr = (nr + rest) / 2
     nrPasi = nrPasi + 1
sf_câttimp
scrie "A fost nevoie de " + nrPasi + " pași"
```

- Codul va afișa mesajele:
  - Numărul: 129
  - Numărul: 65
  - A fost nevoie de 2 pasi

Exemplu 3

Exemplu 3

```
nr, i: întreg
nr = 7
pentru i = 1, nr, 1 execută
    verif: întreg
    verif = i
    dacă verif mod 3 == 1 atunci
        scrie verif
    sf_dacă
sf_pentru
scrie "Ultima valoare verificată: " + verif
```

Exemplu 3

Exemplu 3

```
nr, i: întreg
nr = 7
pentru i = 1, nr, 1 execută
    verif: întreg
    verif = i
    dacă verif mod 3 == 1 atunci
        scrie verif
    sf_dacă
sf_pentru
scrie "Ultima valoare verificată: " + verif
```

- Eroare de compilare pentru că variabila *verif* nu este accesibilă în afara ciclului pentru.
- Codul va afișa numerele 1 și 4, după care va da un mesaj de eroare pentru că variabila *verif* nu este vizibilă în afara ciclului pentru.

Exemplu 4

Exemplu 4

nr, index: întreg
nr = 7
index = 10
câttimp index > nr **execută**
    nr = nr - 1
    suma: întreg
    suma = suma + nr
    scrie "Suma curenta:" + suma
sf_câttimp

Exemplu 4

Exemplu 4

nr, index: întreg
nr = 7
index = 10
câttimp index > nr **execută**
  nr = nr - 1
  suma: întreg
  suma = suma + nr
  scrie "Suma curenta:" + suma
sf_câttimp

- Problema este că variabila sumă nu este inițializată, deci nu putem calcula cât este suma + nr
- Anumite limbaje de programare dau o eroare în acest caz (la compilare sau la rulare), în alte limbaje suma poate fi inițializată cu ceva valoare, dar nu știm ce.

Exemplu 5

Exemplu 5

```
nr,i: întreg
nr = 9
pentru i = 1, nr, 3 execută
    suma: întreg
    suma = 0 + i
    scrie "Suma curenta:" + suma
sf_pentru
suma: întreg
suma = nr * 2
scrie "Suma intermediara: " + suma
pentru i = 1, 4, 1 execută
     suma = suma + i
sf_pentru
scrie "Suma finala: " + suma
```

- Codul de mai sus va afișa următoarele mesaje:
  - Suma curenta: 1
  - Suma curenta: 4
  - Suma curenta: 7
  - Suma intermediara: 18
  - Suma finala: 24