Tipuri Structurate de Date

Consultatii Concurs Mate-Info 2017
(10 Decembrie 2016)

Lector dr. Camelia Chisalita-Cretu
Lector dr. Andreea Vescan

Cuprins

I. Agenda zilei

9:00 - 10:15 - Informatica - Partea I

- Tipuri structurate de date definite de utilizator: analiză şi proiectare

10:15 - 10:30 - Pauza

10:30 - 11:45 - Informatica - Partea a II-a

- Tipuri structurate de date definite de utilizator: analiză şi proiectare

11:45 - 12:00 - Pauza

12:00 - 14:45 - Matematica

II. Informatii generale

Concursul Mate-Info UBB, ediţia a 5-a, 2017

- Data concursului: **8 Aprilie 2017**

- Link: http://www.cs.ubbcluj.ro/concursul-mate-info-ubb/
- Link Tematica: http://www.cs.ubbcluj.ro/admitere/nivel-licenta/tematica-admitere-facultate/

III. Tipuri structurate de date

- O data structurata:
  - Elemente componente
  - O structura - defineste modul de aranjare sau relatiile care se pot stabilii intre elemente.

Observatie. In incercarea de a grupa datele dupa caracteristici comune a fost introdusa notiunea de tip.

- Tip de data = (multime de valori, multime de operatii)
  - Daca valorile din multime sunt atomice ⇒ tip de date atomic
  - Daca valorile din multime sunt structurate ⇒ tip de date structurate
- Structura de date - inregistrare (record)
  - Un obiect descris prin atribute
  - Atributele servesc la descrierea clasei generale de obiecte.
  - Un obiect particular de acest tip este definit prin asocierea unor valori particulare fiecarui atribut.
- Exemplu: Agenda telefonica - nume, adresa, telefon,
  - O persoana din agenda: Maria, Castanilor, 0264010101

IV. Problema 1. Jucarii

Scrieti o aplicatie care il ajuta pe Mos Craciun sa tina evidenta:

- **Jucariilor: idJucarie, denumireJucarie, dimensiuneJucarie**
- **Copiilor: idCopil, numecopil, adresaCopil**
- **Cadourilor: idCadou, idJucarie, idCopil, an**

Si va gestiona o lista de jucarii, o lista de copii si o lista de cadouri cu urmatoarele functionalitati:

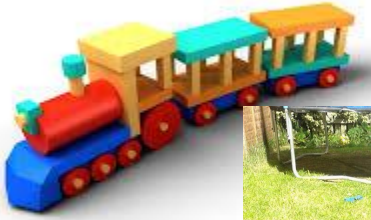
- ❑ **Adauga, elimina, actualizare jucarie/copil**
- ❑ **Cautare jucarie/copil**
- ❑ **Creeaza statistici**
  1. **Jucariile cu dimensiunea mai mare decat o dimensiune precizata;**
  2. **Cea mai mica jucarie de un anumit fel (denumire);**
  3. **Toate cadourile pentru un copil dat**
  4. **Toti copiii care primesc aceeasi jucarie data**
  5. **Lista cadourilor dintr-un an dat.**
  6. **Toate jucariile ce sunt livrate la o adresa data.**

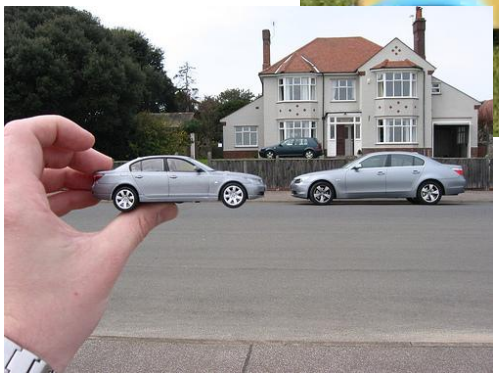
Observatie: Se rezolva in cadrul consultatiilor doar pentru Jucarie/Sir de jucarii. Rezolvarea in totalitate ulterior.

IV. Problema 1. Analiza. Proiectare. Implementare

- **Analiza problema. Tipuri de entitati**
- **Exemplu**
- **Proiectare**
  - **Identificare entitati**
  - **Identificare operatii**
- **Implementare**

IV. Problema 1. Tipuri de Jucarii. Generalitati





IV. Problema 1. Exemplu

Sir de jucarii:

- (1, “minge”, 7.5);
- (2, “*camion*”, 8.5);
- (3, “papusa”, 5);
- (4, “tamburina”, 15);
- (5, “trenulet”, 40);
- (6, “cub magic”, 5);
- (7, “masina”, 10);
- (8, “*camion*”, 6);
- (9, “calut”, 15);
- (10, “papusa”, 10).

- **Statistica 1:**
  - jucariile cu **dimesiune > 9.00**
    - (4, “tamburina”, 15);
    - (5, “trenulet”, 40);
    - (7, “masina”, 10);
    - (9, “calut”, 15);
    - (10, “papusa”, 10);
- **Statistica 2:**
  - Cel mai mic **camion**
    - (8, “camion”, 6);

IV. Problema 1. Proiectare. Entitati si Operatii

A. Identificarea entitatilor:

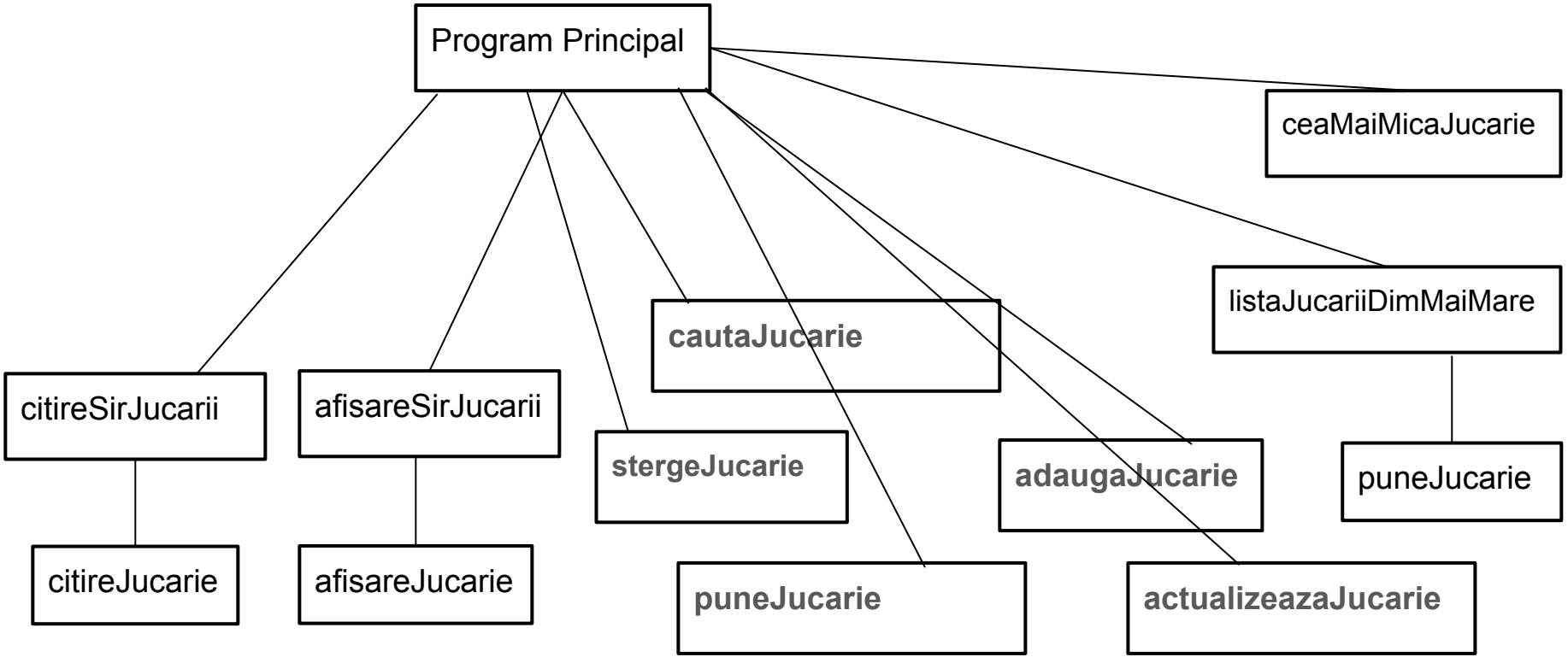
- **Jucarie:**
  - **idJucarie: intreg;**
  - **denumireJucarie: string**
  - **dimensiuneJucarie: real;**

- **SirJucarii:**
  - **sir: Jucarie[ ];**
  - **n: intreg;**

B. Descrierea operatiilor pentru entitati

- **Jucarie:**
  - creare, distrugere;
  - modificare valorii unui atribut, etc.
  - conversie la Jucarie de la un alt tip de date si invers;
  - intrare/iesire;
- **SirJucarii:**
  - creare, distrugere;
  - adaugare, eliminare Jucarie din sir;
  - cautare Jucarie in sir;
  - intrare/iesire;

IV. Problema 1. Diagrama de programare



IV. Problema 1. Operatii entitatea Jucarie

Jucarie - citireJucarie

Date: -

Preconditie:

- True

Rezultate:

- jucarie

Postconditie:

- jucarie ∈ Jucarie

Jucarie

Subalgoritm **citireJucarie** (Jucarie jucarie)

```
    Tipareste (“Introduceti id jucarie: ”);
    Citeste (jucarie.id);

    Tipareste (“Introduceti denumirea jucariei: ”);
    Citeste (jucarie.denumire);

    Tipareste (“Introduceti dimensiunea jucariei: ”);
    Citeste (jucarie.dimensiune);
```

SfSubalgoritm

IV. Problema 1. Operatii entitatea Jucarie

Jucarie - afisareJucarie

Date:

- jucarie

Preconditie:

- jucarie ∈ Jucarie

Rezultate: -

Postconditie:

- s-au afisat atributele jucariei (id, denumire si dimensiune)

Jucarie

Subalgoritm **afisareJucarie** (Jucarie jucarie)

Tipareste (“Id jucarie: ”);

Tipareste (jucarie.id);

Tipareste (“ denumirea jucariei: ”);

Tipareste (jucarie.denumire);

Tipareste (“Dimensiune jucariei: ”);

Tipareste (jucarie.dimensiune);

SfSubalgoritm

IV. Problema 1. Operatii entitatea SirJucarii

SirJucarii - citireSIrJucarii

Date: -

Preconditie:

- True

Rezultate:

- jucarii

Postconditie:

- jucarii ∈ SirJucarii
- jucarii contine n jucarii

SirJucarii

Subalgoritm **citireSirJucarii** (SirJucarii jucarii)

    Tipareste (“Introduceti nr. de jucarii: ”);

    Citeste (jucarii.n);

    Pentru i ← 1, jucarii.n, 1 executa

        citireJucarie (jucarii[i]);

    SfPentru

SfSubalgoritm

IV. Problema 1. Operatii entitatea SirJucarii

SirJucarii - afisareSirJucarii

Date:

- jucarii

Preconditie:

- jucarii ∈ SirJucarii

Rezultate: -

Postconditie:

- s-au afisat atributele jucariilor din sirul *jucarii*

SirJucarii

Subalgoritm **afisareSirJucarii** (SirJucarii jucarii)

    Tipareste (“Sirul este jucarii este: ”);

    Pentru i ← 1, jucarii.nr, 1 executa

        afisareJucarie (jucarii[i]);

    SfPentru

SfSubalgoritm

IV. Problema 1. Operatii entitatea SirJucarii

SirJucarii - cautaJucarie

Date:

- jucarii, id

Preconditie:

- jucarii ∈ SirJucarii
- id: intreg

Rezultate:

- jucarie sau NIL

Postconditie:

- daca (∄ jucarie ∈ jucarii, jucarie.id = id)
  - atunci NIL
  - altfel jucarie

SirJucarii - cauta jucarie dupa id

```
Functia cautaJucarie (SirJucarii jucarii, int id): Jucarie
        i ← 1; gasit ← false;
        CatTimp ((i <= jucarii.n) && (! gasit)) executa
                Daca (jucarii.sir[i].id = id)
                        atunci gasit ← true;
                                jucarie ← jucarii.sir[i];
                        altfel i ← i + 1;
                SfDaca
        SfCatTimp
        Daca (gasit)
                atunci cautaJucarie ← jucarie;
                altfel cautaJucarie ← NIL;
        SfDaca;
SfFunctie
```

IV. Problema 1. Operatii entitatea SirJucarii

SirJucarii - puneJucarie

Date:

- jucarii, jucarie, index

Preconditie:

- jucarii ∈ SirJucarii
- jucarie ∈ Jucarie
- index : intreg, pozitie valida din jucarii.sir

Rezultate:

- -

Postconditie:

- jucarii.sir[index] ← jucarie

SirJucarii - copiaza informatiile despre o jucarie pe o anumita pozitie in sirul cu jucarii

Subalgoritm **puneJucarie** (SirJucarii jucarii, Jucarie jucarie, int index)

    jucarii.sir[index].id ← jucarie.id;
    jucarii.sir[index].denumire ← jucarie.denumire;
    jucarii.sir[index].dimensiune ← jucarie.dimensiune;

SfSubalgoritm

IV. Problema 1. Operatii entitatea SirJucarii

SirJucarii - adaugaJucarie

Date:

- jucarii, jucarie

Preconditie:

- jucarii ∈ SirJucarii
- jucarie ∈ Jucarie

Rezultate:

- jucarii

Postconditie:

- jucarii ∈ SirJucarii
- jucarie a fost adaugat la sirul cu jucarii, daca nu exista o jucarie cu acelasi id

SirJucarii - adaugare jucarie

```
Subalgoritm adaugaJucarie (SirJucarii jucarii,
                           Jucarie jucarie)

    gasit ← cautaJucarie (jucarii, jucarie.id);
    Daca (gasit = NIL) atunci
        jucarii.n ← jucarii.n +1;
        puneJucarie (jucarii, jucarie, n);
SfSubalgoritm
```

IV. Problema 1. Operatii entitatea SirJucarii

SirJucarii - stergeJucarie

Date:

- jucarii, id

Preconditie:

- jucarii ∈ SirJucarii
- id: intreg

Rezultate:

- true/false

Postconditie:

- daca (∄ jucarie ∈ jucarii, jucarie.id = id)
  - atunci false
  - altfel true; jucarie a fost stearsa din sirul cu jucarii

SirJucarii - sterge o jucarie identificata prin id

```
Functia stergeJucarie (SirJucarii jucarii, int id): boolean
        i ← 1; gasit ← false;
        CatTimp ((i <= jucarii.n) && (! gasit)) executa
                Daca (jucarii.sir[i].id = id)
                        atunci gasit ← true;
                                Pentru j ← i+1, jucarii.n, 1,
                                        executa
                                         puneJucarie (jucarii,
                                                        jucarii.sir[j], j-1);
                                SfPentru
                                jucarii.n ← jucarii.n - 1;
                        altfel i ← i + 1;
                SfDaca
        SfCatTimp
        stergeJucarie ← gasit;
SfFunctie
```

IV. Problema 1. Operatii entitatea SirJucarii

SirJucarii - actualizeazaJucarie

Preconditie:

- jucarii este un sir de jucarii;
- id este identificatorul (unic) jucariei cautate

Postconditie:

- **actJucarie** = jucarie din sir, daca exista o jucarie cu id-ul cautat in sir se actulizeaza valorile pentru denumire si dimensiune;

SirJucarii - actulizeaza jucarie dupa id

```
Subalgoritm actualizeazaJucarie (SirJucarii jucarii,
int id, str den, str dim)
        i ← 1; gasit ← false;
        CatTimp ((i <= jucarii.n) && (! gasit)) executa
                Daca (jucarii.sir[i].id = id) atunci
                        jucarii.sir[i].denumire ← den;
                        jucarii.sir[i].dimensiune ← dim;
                        gasit ← true;
                altfel i ← i + 1;
                SfDaca
        SfCatTimp
SfSubalgoritm
```

IV. Problema 1. Statistica 1

Functia listaJucariiDimMaiMare

Date:

- jucarii, D

Preconditie:

- jucarii ∈ SirJucarii
- D: real

Rezultate:

- listaJucarii

Postconditie:

- daca (∀ jucarie ∈ jucarii, jucarie.dimensiune > D)
    - atunci listaJucarii ’= listaJucarii + jucarie

Statistica 1: lista jucariilor cu dimensiune mai mare decat o dimensiune D precizata

Functia **listaJucariiDimMaiMare**(SirJucarii jucarii, real D): SirJucarii

```
    listaJucarii.n ← 0;
    Pentru i ← 1, jucarii.n, 1 executa
        Daca (jucarii.sir[i].dimensiune > D) atunci
            listaJucarii.n ← listaJucarii.n + 1;
            pune (listaJucarii, jucarii.sir[i], listaJucarii.n);
        SfDaca
    SfPentru
    listaJucariiDimMaiMare ← listaJucarii;
SfFunctie
```

IV. Problema 1. Statistica 2

Functia ceaMaiMicaJucarie

Date:

- jucarii, denumire

Preconditie:

- jucarii ∈ SirJucarii
- denumire: string

Rezultate:

- jucarie sau NIL

Postconditie:

- daca (∃ jucarie ∈ jucarii, jucarie.denumire = denumire, jucarie.dimensiune minima)
  - atunci jucarie
  - altfel NIL

Statistica 2: cea mai mica jucarie de un anumit fel (denumire)

```
Functia ceaMaiMicaJucarie (SirJucarii jucarii, string denumire):
Jucarie
Daca (jucarii.n = 0) atunci
        gasit ← false;
altfel  jucarie ← jucarii.sir[0];
        gasit ← false;
        Pentru i ← 1, jucarii.n, 1 executa
                Daca (jucarii.sir[i].denumire = denumire) atunci
                        gasit ← true;
                         Daca (jucarii.sir[i].dimensiune <
                                 jucarie.dimeniune) atunci
                                jucarie ← jucarii.sir[i];
                        SfDaca
                SfDaca
        SfPentru
SfDaca
Daca (gasit) atunci
        ceaMaiMicaJucarie ← jucarie;
altfel ceaMaiMicaJucarie ← NIL;
SfDaca
SfFunctie
```

IV. Problema 1. Algoritmul problemei

Algoritmul Jucarii -

- Statistica 1;
- Statistica 2;

Date:

- jucarii, dim, denumire

Preconditie:

- jucarii ∈ SirJucarii
- dimensiune: intreg;
- denumire :string;

Rezultate:

- lista, jucarie

Postconditie:

- se afiseaza statisticile

```
Algoritmul Jucarii
        citireSirJucarii (jucarii);
        afisareSirJucarii (jucarii);

        Tipareste (“Dati dimensiunea: ”);
        Citeste (dimensiune);

        lista ← listaJucariiDimMaiMare (jucarii, dimensiune);
        Tipareste (“Lista cu dimensiune mai mare decat ”, dimensiune , “este ”);
        afisareSirJucarii (lista);

        Tipareste (“Dati denumirea: ”);
        Citeste (denumire);

        jucarie ← ceaMaiMicaJucarie(jucarii, denumire);
        Daca (jucarie <> NIL) atunci
                Tipareste (“Cea mai mica jucarie cu denumirea”, denumire,  “este ”);
                afisareJucarie (jucarie);
        altfel Tipareste (“Nu exista jucarii cu denumirea”, denumire);
        SfDaca

SfSubalgoritm
```

V. Problema 2. Emotii

Scrieti o aplicatie care gestioneaza emotional persoanele dintr-o incapare:

- **Emotie (expresie gura): fericit, trist, surprins, nervos**
- **Persoana care exprima o emotie**
- **Sir de persoane**

Si va gestiona o lista de persoane cu urmatoarele functionalitati:

a) Sa se determine (iterativ si recursiv) din fiecare tip de emotie numarul de persoane care exprima emotia (frecventa emotiei).
b) Sa se determine emotia predominanta din incapere (emotia cu frecventa cea mai mare).
c) Sa se determine toate subsirurile cu o proprietate data (“fericit” are la stanga si la dreapta lui emotie “trist”) astfel incat distanta dintre proprietati sa fie mai mica decat o valoare data.

Alte functionalitati (Tema):

d) Sa se determine daca exista persoane vecine care au emotii distincte.

e) Sa se insereze intre oricare doua persoane triste, o persoana vesela (aceeasi persoana).

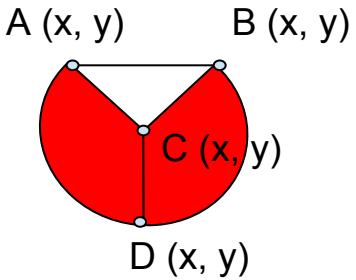
f) Sa se trimita la terapie persoanele nervoase (eliminare din sir a persoanele nervoase).

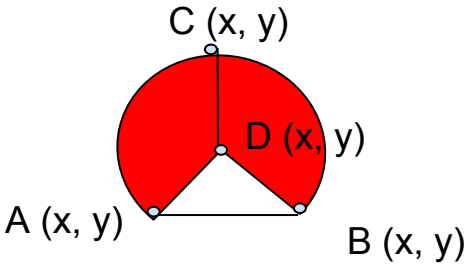
g) Sa se aranjeze persoanele astfel încât cele triste sa nu fie vecine (sau cele nervoase sa nu fie langa cele triste).

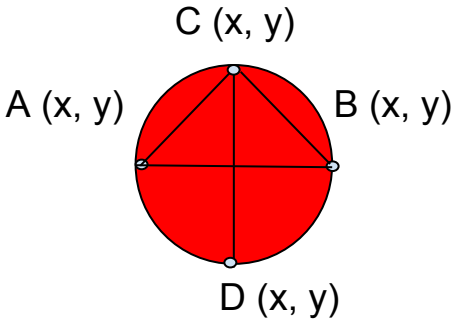
V. Problema 2. Analiza. Proiectare. Implementare

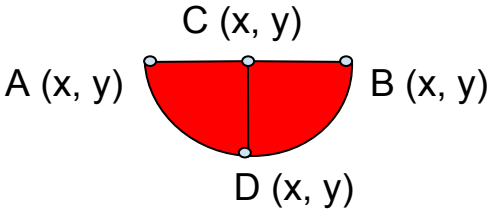
- **Analiza problema. Tipuri de emotii. Reprezentare emotie**
- **Exemplu**
- **Proiectare**
  - **Identificare entitati**
  - **Identificare operatii**
- **Implementare**

V. Problema 2. Tipuri de Emotii. Generalitati









Fericit

Trist

Surprins

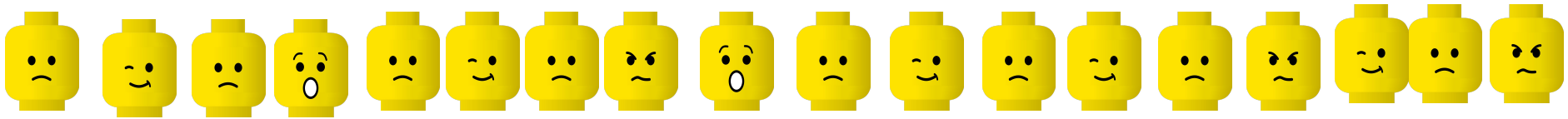
Nervos

V. Problema 2. Exemplu. Cerinta a)

Emotii

Fericit Trist Surprins Nervos

a) Sa se determine (iterativ si recursiv) din fiecare tip de emotie numarul de persoane care exprima emotia (frecventa emotiei).



Numarul de persoane (18 in total):

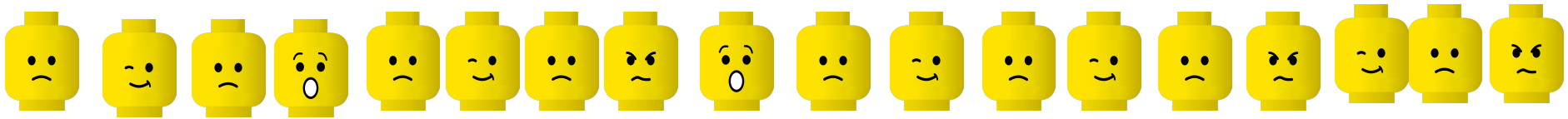
- **fericite (5);**
- **triste (8);**
- **surprinse (2);**
- **nervoase(3).**

V. Problema 2. Exemplu. Cerinta b)

Emotii

Fericit Trist Surprins Nervos

b) Sa se determine emotia predominanta din incapere (emotia cu frecventa cea mai mare).

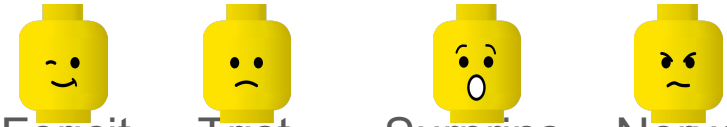


Numarul de persoane (18 in total): fericite (5), triste (8), surprinse (2), nervoase (3)

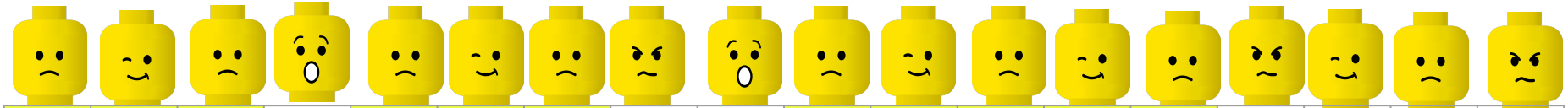
- **Emotia predominanta: tristetea.**

V. Problema 2. Exemplu. Cerinta c)

Emotii



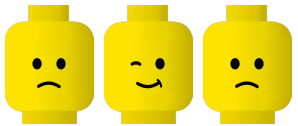
c) Sa se determine toate subsirurile cu o proprietate data (“fericit” are la stanga si la dreapta lui emotie “trist”) astfel incat distanta dintre proprietati sa fie mai mica decat o valoare data.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	F	T	S	T	F	T	N	S	T	F	T	F	T	N	F	T	N

Subsiruri cu prag<=1 si triplet (trist, fericit, trist).

- **Triplete si Subsiruri: (poz 1 la poz 3), (poz 5 la poz 7) , (poz 1 la poz 7), (poz 10 la poz 12), (poz 12 la poz 14), (poz 10 la poz 14).**



V. Problema 2. Proiectare. Entitati si Operatii

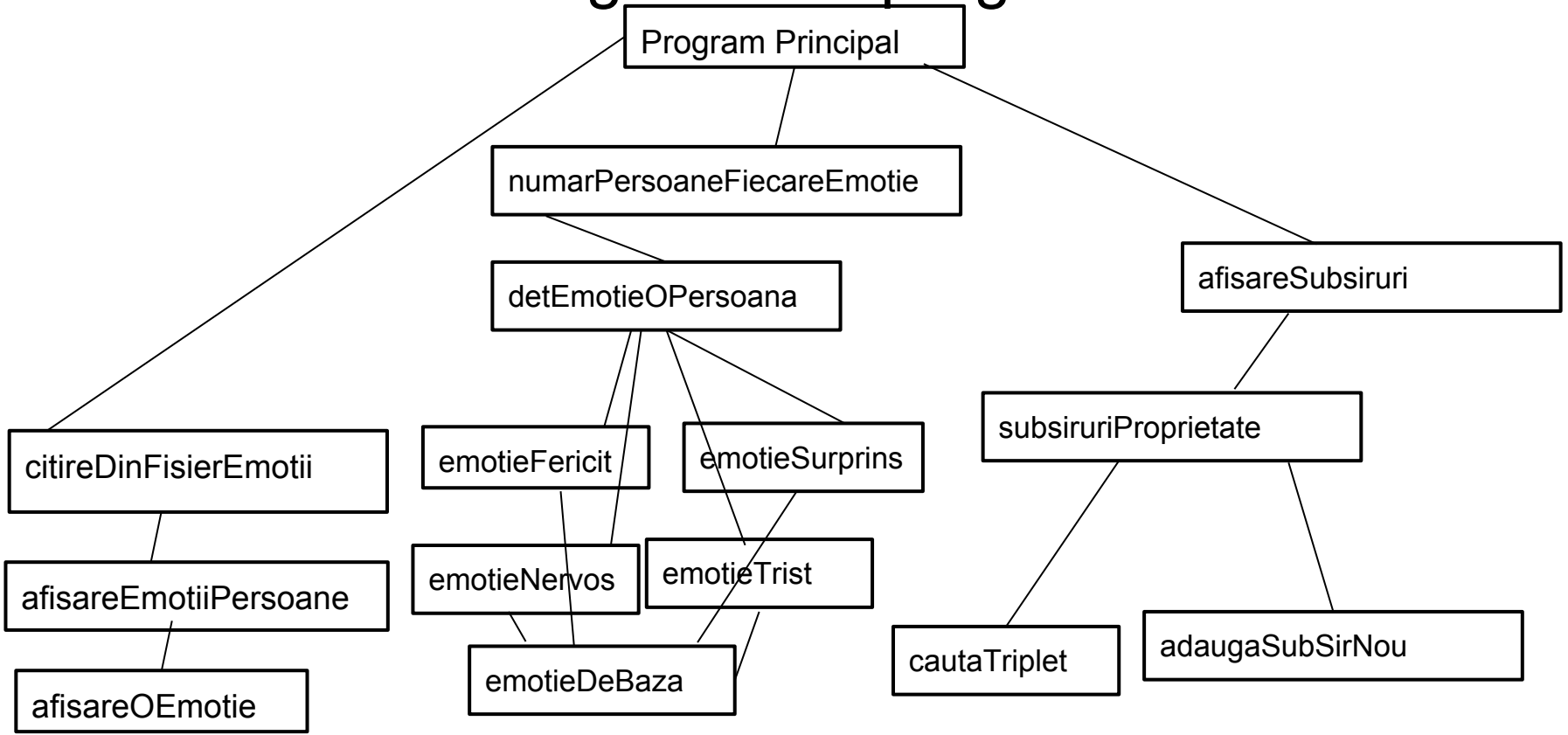
A. Identificarea entitatilor:

- **Punct:**
  - **x,y: intreg;**
- **Emotie**
  - **A, B, C, D: Punct;**
- **SirEmotii:**
  - **nE:intreg**
  - **sirE:Emotie[ ]**
- **PozSubsir**
  - **pS, pF:intreg**
- **SirPozSubsir**
  - **nE:intreg**
  - **sirPozSubsir: PozSubsir[ ]**

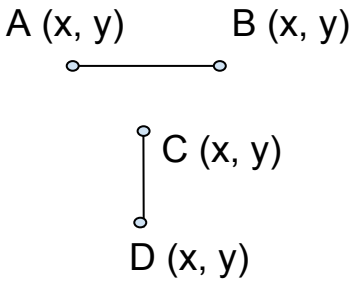
B. Descrierea operatiilor pentru entitati

- citireDinFisierEmotii
- afisareEmotiiPersoane
  - afisareOEmotie
- numarPersoaneFiecareEmotie
  - detEmotieOPersoana
  - emotieFericit
  - emotieSurprins
  - emotieNervos
  - emotieTrist
  - emotieDeBaza
- afisareSubsiruri
  - subsiruriProprietate
  - cautaTriplet
  - adaugaSubSirNou

V. Problema 2. Diagrama de programare



V. Problema 2. Tipuri de Emotii. Generalitati. Implementare



Caracteristici generale ale unei emotii:

- A.y = B.y
- C.x = D.x
- A.x < B.x

Functia emotieDeBaza

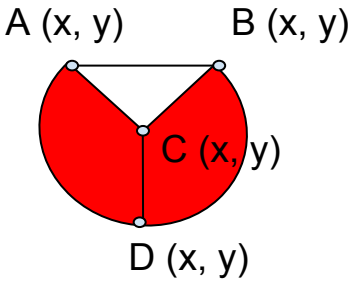
- verifica daca emotia data indica o stare valida

Date: e; **Preconditie:** e ∈ Emotie;
Rezultate: true/false
Postconditie:

- daca (*e este o emotie valida*)
  - atunci true
  - altfel false

```
Functia emotieDeBaza (emotie e): boolean
        eBaza ← false;
        Daca ((e.a.y == e.b.y) &&
                (e.c.x == e.d.x) &&
                (e.a.x < e.b.x)) atunci
            eBaza ←  true;
        SfDaca;
        emotieDeBaza ← eBaza;
SfFunctie
```

V. Problema 2. Tipuri de Emotii. Fericit. Implementare



Particularitate Emotie Fericit:

- **C.y < A.y**
- **D.y < C.y**

Functia emotieFericit

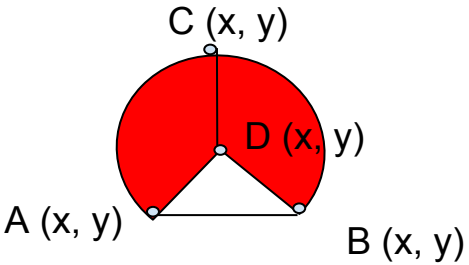
- verifica daca o emotie indica starea “Fericit”

Date: e; **Preconditie:** e ∈ Emotie;
Rezultate: true/false
Postconditie:

- daca (*e este o emotie valida si are particularitatile “Fericit”*)
  - atunci true
  - altfel false

```
Functia emotieFericit (emotie e): boolean
        eF ← false;
        Daca ((emotieDeBaza(e)) &&
            (e.c.y < e.a.y) &&
            (e.d.y < e.c.y)) atunci
                eF ← true;
        SfDaca;
        emotieFericit ← eF;
SfFunctie
```

V. Problema 2. Tipuri de Emotii. Trist. Implementare



Particularitate Emotie Trist:

- **C.y > A.y**
- **D.y > A.y**
- **D.y < C.y**

Functia emotieTrist

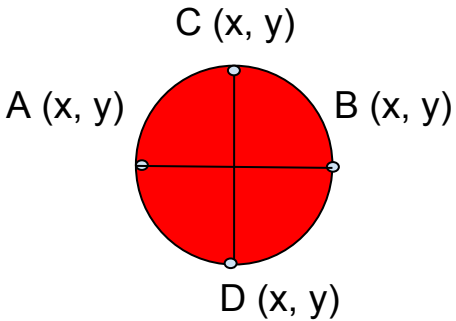
- verifica daca o emotie indica starea “Trist”

Date: e; **Preconditie:** e ∈ Emotie;
Rezultate: true/false
Postconditie:

- daca (*e este o emotie valida si are particularitatile “Trist”*)
  - atunci true
  - altfel false

```
Functia emotieTrist (emotie e): boolean
        eT ← false;
        Daca((emotieDeBaza(e)) &&
                (e.c.y > e.a.y) &&
                (e.d.y>e.a.y) && (e.d.y < e.c.y)) atunci
            eT ← true;
        SfDaca
        emotieTrist ← eT;
SfFunctie
```

V. Problema 2. Tipuri de Emotii. Surprins. Implementare



Particularitate Emotie Surprins:

- **C.y > A.y**
- **D.y < C.y**

Functia emotieSurprins

- verifica daca o emotie indica starea “Surprins”

Date: e; **Preconditie:** e ∈ Emotie;

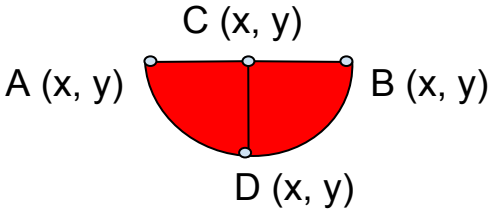
Rezultate: true/false

Postconditie:

- daca (*e este o emotie valida si are particularitatile “Surprins”*)
  - atunci true
  - altfel false

```
Functia emotieSurprins (emotie e): boolean
        eS ← false;
        Daca ((emotieDeBaza(e)) &&
                (e.c.y > e.a.y) &&
                (e.d.y < e.a.y)) atunci
            eS ← true;
        SfDaca
        emotieSurprins ← eS;
SfFunctie
```

V. Problema 2. Tipuri de Emotii. Nervos. Implementare



Particularitate Emotie Nervos:

- **C.y = A.y**
- **D.y < C.y**

Functia emotieNervos

- verifica daca o emotie indica starea “Nervos”

Date: e; **Preconditie:** e ∈ Emotie;
Rezultate: true/false
Postconditie:

- daca (*e este o emotie valida si are particularitatile “Nervos”*)
  - atunci true
  - altfel false

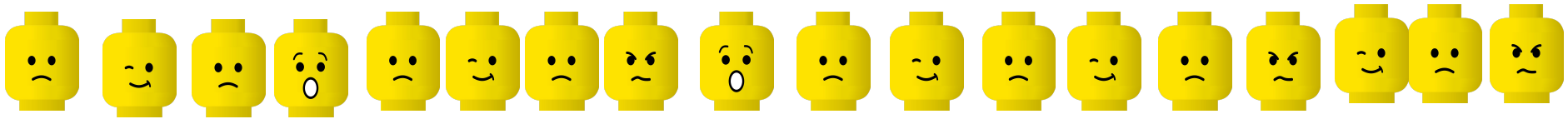
```
Functia emotieNervos (emotie e): boolean
        eN ← false;
        Daca ((emotieDeBaza(e)) &&
                (e.c.y = e.a.y) &&
                (e.d.y < e.c.y)) atunci
            eN ← true;
        SfDaca;
        emotieNervos ← eN;
SfFunctie
```

V. Problema 2. Exemplu. Cerinta a)

Emotii

Fericit Trist Surprins Nervos

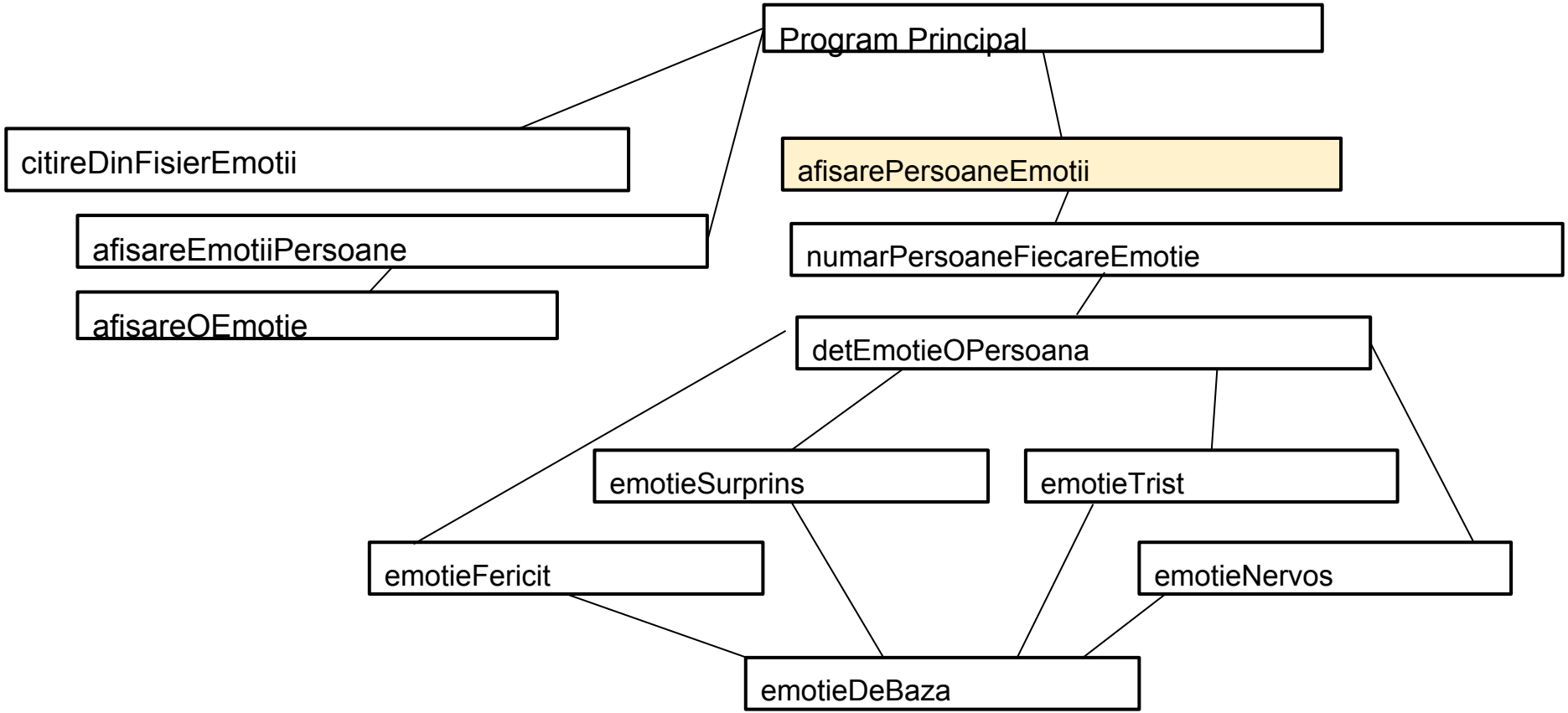
a) Sa se determine (iterativ si recursiv) din fiecare tip de emotie numarul de persoane care exprima emotia (frecventa emotiei).



Numarul de persoane (18 in total):

- **fericite (5);**
- **triste (8);**
- **surprinse (2);**
- **nervoase(3).**

V. Problema 2. Diagrama de programare. Cerinta a)



V. Problema 2. Cerinta a)

Functia detEmotieOPersoana - determina tipul de emotie al unei persoane (Fericit, Trist, Surprins, Nervos);

Date:

- e

Preconditie:

- e ∈ Emotie

Rezultate:

- stare

Postconditie:

- stare ∈ {“fericit”, “trist” “surprins”, “nervos”, “”}

```
Functia detEmotieOPersoana(emotie e): string
  stare ← “”;
  Daca (emotieFericit(e)) atunci
      stare ← "fericit";
   altfel Daca (emotieSurprins(e)) atunci
              stare ← "surprins";
          altfel  Daca (emotieTrist(e)) atunci
                      Stare ← "trist";
                  altfel  Daca (emotieNervos(e)) atunci
                              stare ← "nervos";
                          SfDaca;
                  SfDaca;
          SfDaca;
  SfDaca;
 detEmotieOPersoana ← stare;
SfFunctie
```

V. Problema 2. Cerinta a)

Functia numarPersoaneFiecareEmotie - determina numarul de persoane din sir care au o emotie data (“Fericit”, “Trist”, “Surprins”, “Nervos”);

Date:

- emotii, emotieS

Preconditie:

- emotii ∈ SirEmotii
- emotieS: ∈ {“fericit”, “trist” “surprins”, “nervos”}

Rezultate:

- nrPersoaneOEmotie

Postconditie:

- *nrPersoaneOEmotie* este numarul de persoane cu emotia *emotieS*

// versiune iterativa

```
Functia numarPersoaneFiecareEmotie(SirEmotii emotii,
string emotieS): intreg
    nrPersoaneOEmotie ← 0;
    Pentru  i ← 0, emotii.nE, 1 executa
      tipEmotie ← detEmotieOPersoana(emotii, emotieS);
      Daca (tipEmotie = emotieS) atunci
            nrPersoaneOEmotie ← nrPersoaneOEmotie+1;
      SfDaca;
    SfPentru;
    numarPersoaneFiecareEmotie ← nrPersoaneOEmotie;
SfFunctie
```

V. Problema 2. Cerinta a)

Functia numarPersoaneFiecareEmotie<u>Recursiv</u> - determina numarul de persoane din sir care au o emotie data (“Fericit”, “Trist”, “Surprins”, “Nervos”);

Date:

- emotii, emotieS, pozitie

Preconditie:

- emotii ∈ SirEmotii
- emotieS: ∈ {“fericit”, “trist”, “surprins”, “nervos”}
- pozitie: intreg

Rezultate:

- nrPersoaneOEmotie

Postconditie:

- *nrPersoaneOEmotie* este numarul de persoane cu emotia *emotieS*

```
//Versiune recursiva

Functia numarPersoaneFiecareEmotieRecursiv
(SirEmotii emotii, string emotieS, intreg pozitie): intreg
      Daca (pozitie = -1) atunci
        numarPersoaneEmotieRecursiv ← 0;
      altfel
        tipEmotie ← detEmotieOPersoana(emotii,
        emotieS);
        Daca (tipEmotie = emotieS) atunci
              1+
              numarPersoaneFiecareEmotieRecursiv
              (emotii, emotieS, pozitie-1);
        altfel numarPersoaneFiecareEmotieRecursiv
              (emotii, emotieS, pozitie-1);
        SfDaca;
SfFunctie

apel:
nrPersoaneFericite ←
      numarPersoaneFiecareEmotieRecursiv
      (emotii, “Fericit”, emotii.nE);
```

V. Problema 2. Cerinta a)

Subalgoritmul afisarePersoaneEmotii - afiseaza numarul de persoane din sir care au un anumit tip de emotie (“Fericit”, “Trist”, “Surprins”, “Nervos”);

Date:

- emotii

Preconditie:

- emotii ∈ SirEmotii

Rezultate:

- -

Postconditie:

- pentru fiecare tip de emotie se afiseaza numarul de persoane din sir care au un atumit tip de emotie

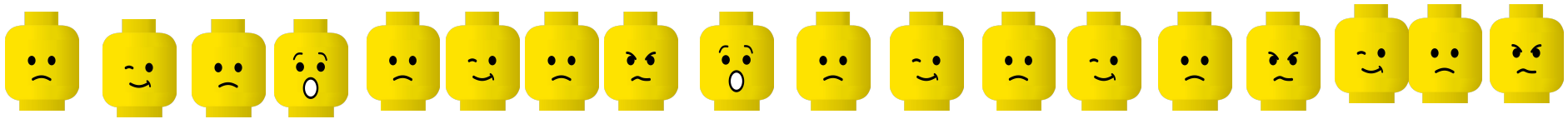
```
Subalgoritmul afisarePersoaneEmotii (SirEmotii emotii)

    Tipareste (“Numarul de persoane fericite este:”);
    Tipareste (numarPersoaneFiecareEmotie (emotii, “Fericit”));

    Tipareste (“Numarul de persoane triste este:”);
    Tipareste (numarPersoaneFiecareEmotie (emotii, “Trist”));

    Tipareste (“Numarul de persoane surprinse este:”);
    Tipareste (numarPersoaneFiecareEmotie (emotii, “Surprins”));

    Tipareste (“Numarul de persoane nervoase este:”);
    Tipareste (numarPersoaneFiecareEmotie (emotii, “Nervos”));

SfSubalgoritm
```

V. Problema 2. Exemplu. Cerinta b)

Emotii

Fericit Trist Surprins Nervos

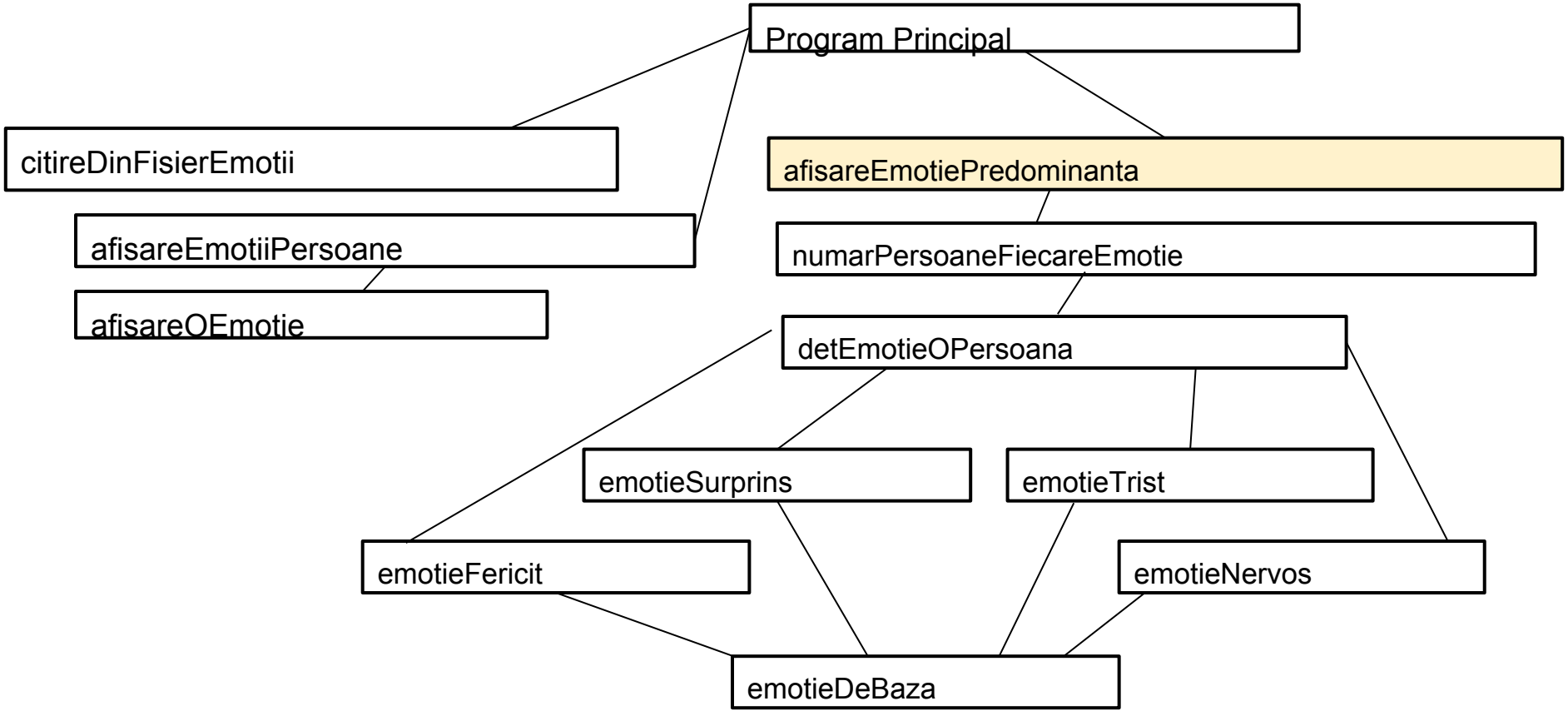
b) Sa se determine emotia predominanta din incapere (emotia cu frecventa cea mai mare).



Numarul de persoane (18 in total): fericite (5), triste (8), surprinse (2), nervoase (3)

- **Emotia predominanta: tristetea.**

V. Problema 2. Diagrama de programare. Cerinta b)



V. Problema 2. Cerinta b)

Subalgoritmul determinaEmotiePredominanta - determina emotia predominanta din sirul de emotii;

Date:

- emotii

Preconditie:

- emotii ∈ SirEmotii

Rezultate:

- ePredominanta, maximPredominant

Postconditie:

- ePredominanta : ∈ {“fericit”, “trist” “surprins”, “nervos”}

Subalgoritmul afiseazaEmotiePredominanta - afiseaza emotia predominanta;

Date:

- ePredominanta, maximPredominant

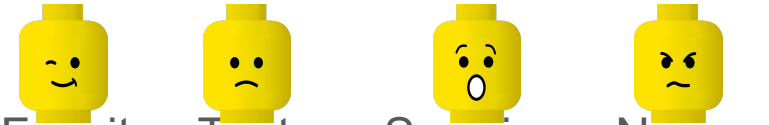
Rezultate:

- -

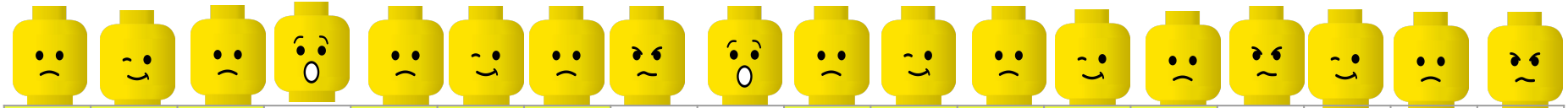
```
Subalgoritmul determinaEmotiePredominanta (SirEmotii emotii, int&
maximPredominant, string ePredominanta)
        nrPersFericite ← numarPersoaneFiecareEmotie (emotii,“Fericit”);
        nrPersTriste ← numarPersoaneFiecareEmotie (emotii,“Trist”);
        nrPersSurprinse ← numarPersoaneFiecareEmotie (emotii,“Surprins”);
        nrPersNervoase ← numarPersoaneFiecareEmotie (emotii,“Nervos”);
        maximPredominant ← nrPersFericite;
        ePredominanta ← “Fericit”;
        Daca (maximPredominant < nrPersTriste) atunci
                maximPredominant ← nrPersTriste;
                ePredominanta ← “Trist”;
        altfel Daca (maximPredominant < nrPersSurprinse) atunci
                maximPredominant ← nrPersSurprinse;
                ePredominanta ← “Surprins”;
            altfel Daca (maximPredominant < nrPersNervoase) atunci
                        maximPredominant ← nrPersNervoase;
                        ePredominanta ← “Nervos”;
                    SfDaca;
            SfDaca;
        SfDaca;
sfSubalgorithm

Subalgoritmul afiseazaEmotiePredominanta (string ePredominanta, int
maximPredominant)
        Tipareste (“Emotia predominanta este: “);
        Tiipareste (ePredominanta);
        Tipareste (“cu numarul maxim de persoane: “);
        Tipareste (maximPredominant);
SfSubalgoritm
```

V. Problema 2. Exemplu. Cerinta c)

Emotii



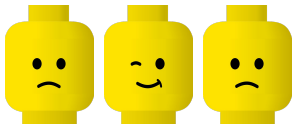
c) Sa se determine toate subsirurile cu o proprietate data (“fericit” are la stanga si la dreapta lui emotie “trist”) astfel incat distanta dintre proprietati sa fie mai mica decat o valoare data.



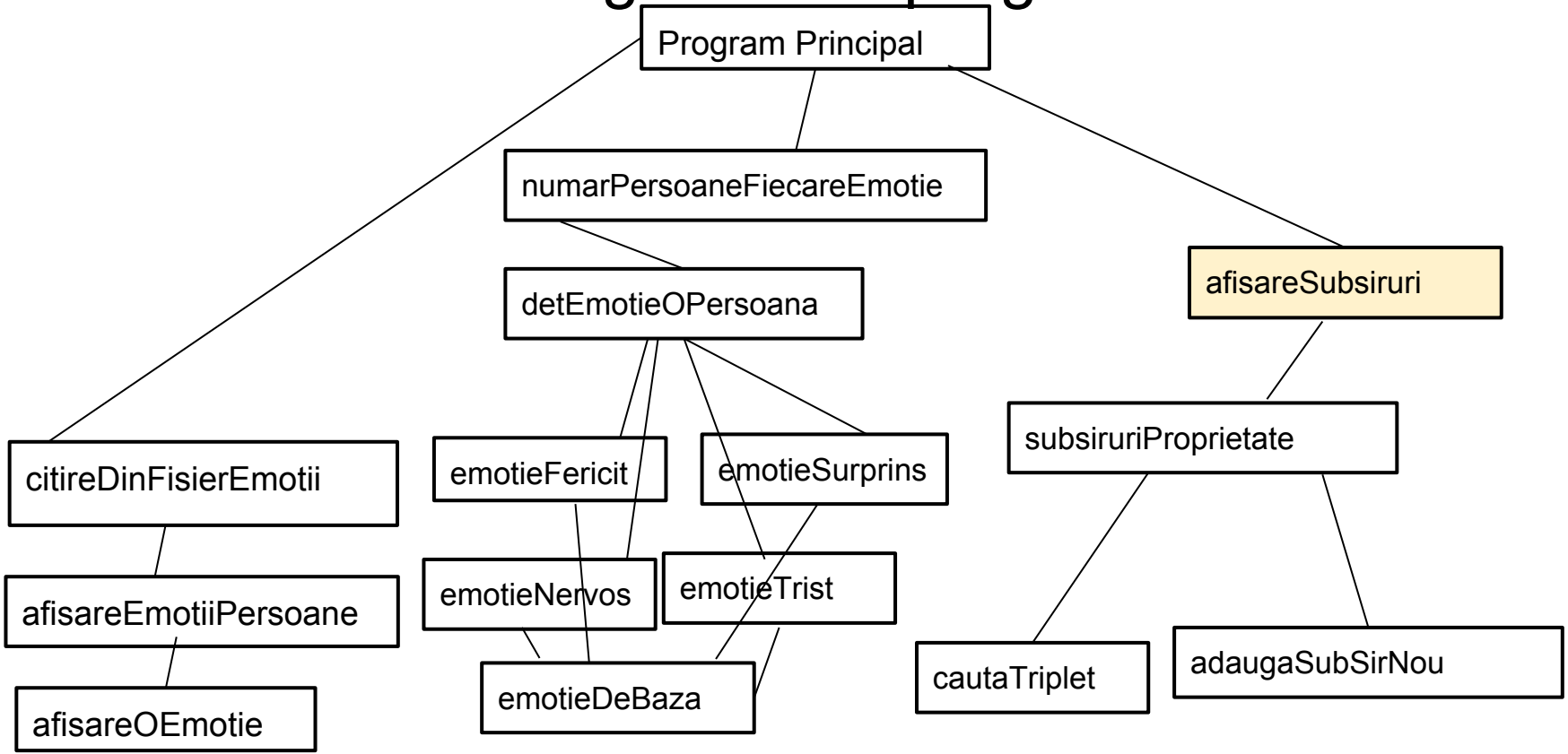
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T	F	T	S	T	F	T	N	S	T	F	T	F	T	N	F	T	N

Subsiruri cu prag<=1 si triplet (trist, fericit, trist).

- **Triplete si Subsiruri: (poz 1 la poz 3), (poz 5 la poz 7) , (poz 1 la poz 7), (poz 10 la poz 12), (poz 12 la poz 14), (poz 10 la poz 14).**



V. Problema 2. Diagrama de programare



V. Problema 2. Cerinta c)

Subalgoritmul cautaTriplet - determina o secventa de emotii incepand cu pozitia data; secventa indeplineste proprietatea ceruta *“fericit” intre doua persoane “triste”*;

Date:

- emotii, pozitie

Preconditie:

- emotii ∈ SirEmotii
- pozitie: intreg;

Rezultate:

- pS, pF

Postconditie:

- pS este pozitia de inceput, iar pF este pozitia de sfarsit a unei secvente din sirul emotii care indeplineste proprietatea ceruta (“fericit” intre doua persoane “triste”)

```
Subalgoritmul cautaTriplet(SirEmotii emotii, intreg, pozitie, intreg
pS, intreg pF)
        pS ← -1; pF ← -1;
        gasit ← false;
        CatTimp ((!gasit) && (pozitie < emotii.nE)) executa
                CatTimp ((pozitie < emotii.nE) &&
                        (!emotieTrist(emotii.sir[pozitie]))) executa
                        pozitie ← pozitie +1;
                SfCatTimp
                Daca ((pozitie < emotii.nE) && (pozitie+2 < emotii.nE))
                        atunci
                        Daca ((emotieFericit(emotii.sir[pozitie+1])) &&
                                (emotieTrist(emotii.sir[pozitie+2])))
                        atunci
                                pS ← pozitie;
                                pF ← pozitie + 2;
                                gasit ← true;
                        altfel pozitie ← pozitie +1;
                        SfDaca;
                altfel pozitie ← pozitie +1;
                SfDaca;
        SfCatTimp;
SfSubalgoritm
```

V. Problema 2. Cerinta c)

Subalgoritmul adaugaSubSirNou - retine pozitia de inceput si de sfarsit a unui nou subsir care indeplineste proprietatea ceruta *“fericit” intre doua persoane “triste”*;

Date:

- subSiruri, pS, pF

Preconditie:

- subSiruri ∈ SirPozSubSir
- pS, pF: intreg;

Rezultate:

- subSiruri

Postconditie:

- subSiruri ’ = subSiruri + (pS, pF)

```
Subalgoritmul adaugaSubSirNou(SirPozSubSir subSiruri,
intreg pSNoua, intreg pFNoua)
        subSiruri.sirPozSubsir[subSiruri.nE].pS ← pSNoua;
        subSiruri.sirPozSubsir[subSiruri.nE].pF ← pFNoua;
        subSiruri.nE ← subSiruri.nE + 1;
SfSubalgoritm
```

V. Problema 2. Cerinta c)

Subalgoritmul subsiruriProprietate(SirEmotii emotii, intreg prag, SirPozSubsir subSiruri)

- determina subsirurile care indeplinesc proprietatea *“fericit” intre doua persoane “triste”* din sirul cu emotii, pentru un prag dat;

prag = indica un numar maxim de pozitii acceptat intre doua secvente de emotii determinate, astfel incat acestea sa formeze un singur subsir; in cazul in care numarul de pozitii dintre secvente este mai mare atunci secventele formeaza subsiruri distincte;

Date: emotii, prag
Preconditie:

- Emotii ∈ SirEmotii;
- subSiruri ∈ SirPozSubSir;
- prag: intreg;

Rezultate:

- subSiruri

Postconditie:

- subSiruri contine toate perechile (pS, pF) pentru toate subsirurile de emotii care respecta proprietatea data, cu pragul maxim dat;

V. Problema 2. Cerinta c)

```
Subalgoritmul subsiruriProprietate (SirEmotii
emotii, intreg prag, SirPozSubsir subSiruri) {

    subSiruri.nE ← 0;
    pS ← -1; pF ← -1;
    pS2 ← -1, pF2 ← -1;
    existaTriplet2 ← false;

    i ← 0;
    cautaTriplet (emotii, i, pS, pF);
    Daca ((pS <> -1) && (pF <> -1)) atunci
        adaugaSubSirNou (subSiruri, pS, pF);
        cautaTriplet (emotii, pF, pS2, pF2);
        Daca ((pS2 <> -1) && (pF2 <> -1)) atunci
            adaugaSubSirNou (subSiruri,
            pS2, pF2);
            existaTriplet2 ← true;
```

```
        CatTimp ((existaTriplet2)){
            Daca ((pS2 - pF-1) <= prag) atunci
                pS ← pS;
                pF ← pF2;
                adaugaSubSirNou(subSiruri, pS, pF);
            altfel
                pS ← pS2;
                pF ← pF2;
                adaugaSubSirNou (subSiruri, pS, pF);
            SfDaca
            cautaTriplet (emotii, pF, pS2, pF2);
            Daca ((pS2 <> -1) && (pF2 <> -1)) atunci
              adaugaSubSirNou (subSiruri, pS2, pF2);
              existaTriplet2 ← true;
            altfel existaTriplet2 ← false;
            SfDaca
        SfCatTimp
    SfDaca
  SfDaca
SfSubalgoritm
```

V. Problema 2. Cerinta c)

Subalgoritmul afisareSubsiruri - afiseaza toate subsirurile care indeplinesc proprietatea ceruta (*“fericit” intre doua persoane “triste”*) din sirul cu emotii, pentru un prag dat;

Date:

- emotii, subSiruri

Preconditie:

- emotii ∈ SirEmotii
- subSiruri ∈ SirPozSubSir

Rezultate:

- -

Postconditie:

- se afiseaza subsirurile de emotii care indeplinesc proprietatea pentru pragul dat

```
Subalgoritmul afisareSubsiruri (SirEmotii emotii, SirPozSubSir
subSiruri)
    Pentru i ← 0, subSiruri.nE-1, 1 executa
        Tipareste (“subsir de la pozitia “);
        Tipareste (subSiruri.sirPozSubsirozSubsir[subSiruri.nE].pS);
        Tipareste (“ pana la pozitia “);
        Tipareste (subSiruri.sirPozSubsirozSubsir[subSiruri.nE].pF);
   SfPentru
SfSubalgoritm
```

V. Problema 2. Algoritmul problemei

Algoritmul Emotii -

- Cerinta a);
- Cerinta b);
- Cerinta c);

Date:

- emotii, prag

Preconditie:

- emotii ∈ SirEmotii
- prag: intreg

Rezultate:

- subsiruri

Postconditie:

- se afiseaza subsirurile de emotii care indeplinesc proprietatea si pragul dat

Algoritmul **Emotii**

citireDinFisierEmotii (emotii);
afisareEmotiiPersoane (emotii);

afisarePersoaneEmotii (emotii);

afisareEmotiePredominanta (emotii);

Tipareste (“Dati pragul maxim: ”);
Citeste (prag);

subsiruriProprietate (emotii, subsiruri, prag);
afisareSubsiruri (emotii, subsiruri);

SfSubalgoritm


Q
A