OOP: Lista problemelor pentru tema de casă

VladG
ver. 1.0

21.11 - 24.12.2016

(1) Pentru următoarea definiţie de clasă:

```
class Temperature
{
  public:
    // Sets the member variables to the values given as arguments.
    double degrees;
    char scale; // <F> for Fahrenheit or <C> for Celsius
    void set( double newDegrees, char newScale);
};
```

Cerinţe obligatorii:

(a) Daţi o definiţie potrivită funcţiei membru `set()`.

(b) În ce măsură este potrivită o funcţie prietenă (`friend`)? Un exemplu.

(c) Supraîncărcaţi un operator considerat util în acest context (e.g. adunare, împărţire).

(d) Compuneţi un scurt program prin care să exemplificaţi alegerile făcute.

(2) Se consideră clasa definită în modul următor:

```
class Point
{
  private:  // Data members (private)
    int x, y;
  public:  // Member functions
    void set(int new_x, int new_y);
    int get_x();
    int get_y();
};
```

Se cer:

(a) Implementaţi funcţiile membre.

(b) Modificaţi funcţia `set()` astfel încât să nu se admită valori mai mari de 100 pentru `x` şi `y`. Pentru valori mai mari de 100 acestea vor fi reduse cu 100. Verificaţi acest comportament şi în funcţia principală.

(c) Modificaţi funcţionalitatea clasei astfel încât să reuşiţi crearea unui vector cu 7 elemente de tipul `Point`.

i. Construiţi o buclă în cadrul căreia să cereţi de la intrare valori pentru fiecare din cele 7 obiecte.

ii. Afişaţi în această buclă valorile citite.

Indicaţie: puteţi folosi numele clasei pentru a construi un vector, la fel cum folosiţi orice alt tip nativ de dată C++.

(3) Supraîncărcarea Operatorilor

Se dă definiţia următoare a unei clase:

```
class Sales_item
{
    friend std::istream& operator>>(std::istream&, Sales_item&);
    friend std::ostream& operator<<(std::ostream&, const Sales_item&);
    friend bool operator<(const Sales_item&, const Sales_item&);
    friend bool operator==(const Sales_item&, const Sales_item&);
  public:
    Sales_item() = default; // default constructor
    Sales_item(const std::string &book) :  bookNo(book) {}
    Sales_item(std::istream &is) { is >> *this; }
  public:
    // operations on Sales_item objects
    // member binary operator:  left-hand operand bound to implicit this pointer
    Sales_item& operator+=(const Sales_item&);
    // operations on Sales_item objects
    std::string isbn() const { return bookNo; }
    double avg_price() const;
  // private members
  private:
    std::string bookNo; // este iniţializat implicit cu şirul vid
    unsigned units_sold = 0; // iniţializat explicit
    double revenue = 0.0;
};
```

Studiind codul prezentat, se cer:

(a) Scrieţi un program prin care să parcurgeţi întreaga mulţime de tranzacţii legate de vânzarea de cărţi. Citiţi datele fiecărei tranzacţii şi afişaţi-le la ecran.

(b) Scrieţi un program în care să citiţi datele a două obiecte `Sales_item` cu acelaşi $ISBN$. Calculaţi suma acestora.

(c) Scrieţi un program care citeşte mai multe tranzacţii pentru acelaşi $ISBN$. Afişaţi *suma* tuturor tranzacţiilor citite anterior.

(4) Funcţionalitatea Constructorului de Copiere. [5]

Presupunem că `Point` este o clasă având un constructor de copiere `public`.

(a) Pentru fragmentul de cod următor identificaţi fiecare situaţie şi justificaţi utilizarea constructorului de copiere în acea situaţie.

```
Point global;
Point foo_bar(Point arg)
{
    Point local = arg;
    Point *heap = new Point(global);
    *heap = local;
    Point pa[4] = { local, *heap };
    return *heap;
}
```

(b) Care este declaraţia corectă a constructorului de copiere al clasei `Point`?

- `friend Point(Point);`
- `Point(const Point&);`

- `void Point::Point(const Point&);`
- `Point& Point(const Point&);`
- `Point Point(Point&);`
- Nici una dintre variantele prezentate (şi atunci sintaxa este: ...)

(5) Search & Replace [1]

Pentru procesarea unui text, una dintre operaţiile uzuale este şi cea de *search and replace*.
În problema de faţă se va considera modul de funcţionare următor: se doreşte executarea unei *serii* de operaţii de înlocuire, pe baza unor *reguli*. O regulă va specifica:

- Şirul care trebuie căutat. Acesta se specifică pe un rând;
- Şirul cu care cel găsit va trebui înlocuit. Acesta se va specifica pe rândul următor.

Cerinţe:

(a) Pentru rezolvarea problemei folosiţi caracteristicile clasei `string` din C++.

(b) Programul va interacţiona cu dispozitivele de intrare/ieşire, astfel:
- Date de intrare:
  - Mulţimea de reguli se va specifica într-un fişier de intrare (e.g. `reguli.in`)
  - Prima linie din fişierul de reguli conţine numărul acestora (N) (e.g. 10). Este întotdeauna pozitiv.
  - Următoarele $2 \cdot N$ linii conţin perechi de şiruri, pe principiul construcţiei unei reguli, enunţat mai sus.
  - Ultimele rânduri ale fişierului conţin textul în care se vor face înlocuirile.
- Date de ieşire:
  - Un fişier de ieşire (e.g. `fisier.out`), ce va conţine doar textul obţinut după aplicarea celor N reguli de înlocuire.

(c) Date fiind regulile şi textul care trebuie prelucrat, să se determine forma finală a textului.

Un exemplu:

`reguli.in`	`fisier.out`
10 atunci acum Astazi Maine ... **atunci** a inceput totul. **Astazi** voi incepe sa analizez efectele. Sper ca echipajul sa nu aiba probleme.	**acum** a inceput totul. **Maine** voi incepe sa analizez efectele. Sper ca echipajul sa nu aiba probleme.

(6) Numere Complexe [1]

Un număr complex se reprezintă algebric prin cele două componente, reală şi imaginară:

$$z = 2.5 - 4 \cdot i$$

Cerinţe:

(a) Definiţi o clasă `Complex` pentru lucrul cu astfel de numere.

(b) Demonstraţi că aplicaţia funcţionează scriind un program pentru aceasta.

Indicaţii:

1. Datele membre sunt partea reală şi imaginară. Recomandarea tipului de dată pentru acestea este `double`.

2. Vor fi definiţi toţi constructorii necesari.
3. Vor fi construiţi operatorii necesari lucrului cu numerele complexe.

(7) Polinoame [1]

Convenim ca reprezentarea unui polinom să o facem prin grad (*grad*) şi coeficienţi. Coeficienţii vor fi reţinuţi într-un vector cu *grad*+1 poziţii. Un coeficient lipsă va fi marcat cu 0 pe poziţia corespunzătoare din vector.
De exemplu, pentru polinomul:

$$P_5(x) = x^5 - 3 \cdot x^3 + 2 \cdot x^2 - 1$$

coeficienţii pot fi stocaţi astfel:

coeficienţi	-1	2	-3	0	1

Cerinţe de lucru:

(a) Construiţi o clasă `Polinom` care să ofere lucrul cu polinoame. Operaţiile impuse sunt:
- Adunare
- Scădere
- Înmulţire
- Împărţire (descriind şi algoritmul folosit).

(b) Scrieţi şi o scurtă aplicaţie prin care să demonstraţi funcţionalitatea clasei.

Observaţii:

(a) Coeficienţii sunt reţinuţi începând cu termenul liber. Nu este însă obligatoriu acest lucru.

(b) Vectorul coeficienţilor *se poate aloca dinamic*. Problema nu specifică acest lucru şi nu se va puncta suplimentar şi nici depuncta absenţa versiunii cu alocare dinamică.

(8) Stivă [2]

Să se implementeze obiectual caracteristicile şi operaţiile tipice unei structuri de date dinamice de tip stivă.
Indicaţie:
Un posibil *tipar* al acestei clase ar putea fi:

```
class Stiva {
  void ** tab;
  int head;
  int capacity;
public:
  Stiva();
  Stiva(int _capacity); // constructor cu parametru
  Stiva(Stiva &); // constructor de copiere
  ~Stiva(); // destructor implicit
  void* pop(); // operaţia de eliminare din stivă a elementului din vârf
  void* peek(); // oferă adresa unde se află stocată valoarea din vârful stivei
  void push(void*); // operaţia de adăugare pe stivă (în vârful acesteia)
  // a unei variabile, a cărei adresă este cunoscută
  int isEmpty(); // poate returna şi tipul bool
  int isFull(); // poate returna şi tipul bool
} ;
```

Observaţii:

a) Tiparul prezentat mai sus este cu titlu *exemplificativ*.

b) Acesta oferă o idee asupra a ceea ce presupune interfaţarea cu o astfel de clasă.

c) Orice alte propuneri sunt acceptate, cu condiţia să respecte algoritmii specifici şi principiile acestui tip de dată dinamic.

(9) Localitate [3]

1) Implementaţi următoarea propunere de *ierarhie de clase* (v. mai jos).
2) Completaţi punctele de suspensie cu ceea ce consideraţi necesar pentru ca ierarhia astfel construită să aibă sens.
3) Testaţi într-un program soluţia propusă.
4) Dacă consideraţi util, faceţi toate modificările dorite, chiar dacă ele rescriu porţiuni din varianta propusă mai jos.

```
class Localitate {
protected:
  char *denumire;
  int cod;
  long nr_locuitori;
public:
  ...
  virtual void display();
};


class Oras : public Localitate {
protected:
  int nrBlocuri;
public:
  ...
};


class Capitala : public Oras {
protected:
  char numePrefect;
public:
  ...
};


class Judet {
  Localitate *p;
  int nrLoc;
public:
  ...
};
```

(10) Bibliotecă [1] *(problemă pentru 2 persoane)*

Realizaţi o aplicaţie care să permită gestiunea unei biblioteci tipice.
Operaţiile specifice minimale vor include:

(a) Evidenţa **cărţilor**:

i. Adăugare carte,
ii. Ştergere carte,
iii. Căutare după
   A. titlu (cu potrivire totală sau parţială,
   B. autor,
   C. domeniu (v. mai jos).

Pentru fiecare carte se vor reţine:

i. Cota cărţii
ii. Titlul complet
iii. Autorul(ii)
iv. Editura
v. Anul apariţiei.

(b) Domeniul de specialitate (e.g. Beletristică, Tehnică, SF, Filosofie, Ştiinţe Umaniste etc.)

(c) Evidenţa **cititorilor**, şi anume:

a) Adăugare cititor,
b) Ştergere cititor,
c) Căutare cititor după nume sau CNP.

Pentru fiecare cititor se vor reţine următoarele date:

i. CNP
ii. Numele complet
iii. Adresa completă
iv. Telefon (obligatoriu)
v. Adresă de e-mail.

(d) Evidenţa **împrumuturilor**, şi anume:

i. Vizualizarea împrumuturilor,
ii. Vizualizarea împrumuturilor restante (împreună cu datele cititorului(ilor)
iii. Vizualizarea împrumuturilor unui cititor.

Pentru fiecare împrumut se vor reţine următoarele date:

i. cota cărţii
ii. codul cititorului
iii. data împrumutului
iv. data returului.

(e) Într-o variabilă `static` veţi reţine *numărul de zile* pentru care poate fi împrumutată o carte (acest număr este fix ⇒ nu variază nici în funcţie de cititor, nici de carte).

(11) Figuri Geometrice [3]

Să se implementeze clasa `FiguraGeometrica`.

1) Din aceasta să se deriveze `Cerc` şi `Poligon`.
2) Să se deriveze din `Poligon` clasele `Patrulater` şi `Triunghi`.
3) Din `Patrulater` să se deriveze clasa `Dreptunghi`.
4) Funcţii `virtuale` impuse: `getAria()`, `getPerimetru()`
5) Scrieţi un program în care să verificaţi funcţionalitatea propusă.

(12) Data Calendaristică

Modelaţi şi exemplificaţi funcţionalitatea clasei `Data`.

Cerinţe de proiectare:

1) Aceasta trebuie să aibă ca atribute private: `An, Luna, Zi`.

2) Implementaţi metodele speciale: `setAn()`, `setLuna()`, `setZi()`, `getAn()`, `getLuna()`, `getZi()` prin care se pot atribui/citi valorile pentru caracteristicile Datei calendaristice.
3) Stabilirea valorilor în metodele mutator `setAn()`, `setLuna()`, `setZi()` se va face cu date introduse de la tastatură.
4) Implementaţi un *constructor implicit* şi un *constructor cu argumente* pentru această clasă.
5) Implementaţi metodele standard:
   - `LunaUrmatoare()`, care afişează aceeaşi dată dar din luna următoare,
   - `Mâine()`, care afişează data completă din ziua următoare.
6) În funcţia `main()` creaţi:
   i. un obiect `D0` care să preia datele din constructorul implicit,
   ii. un al doilea obiect `D1` care preia datele din mutatori,
   iii. un treilea obiect `D2` care preia datele din constructorul cu argumente.
7) Afişati caracteristicile obiectului `D2`.
8) Aplicaţi metoda `LunaUrmatoare()` pentru obiectul `D0`, metoda `Mâine()` pentru obiectul `D0`.

Observaţii:

1) Toate metodele şi constructorii trebuie *declaraţi în interiorul* clasei şi *definiţi în exteriorul* acesteia.
2) Comentaţi codul scris. Veţi plasa câte un scurt comentariu explicativ pentru fiecare atribut, funcţie, constructor, setter, getter, obiecte, apel de metode etc.

(13) VECTOR [2]

Să se implementeze clasa următoare ce reprezintă **tipul vector** împreună cu operaţiile specifice acestuia. Ţinând cont de implementarea propusă scrieţi un scurt program, în care să verificaţi funcţionalitatea propusă.
Indicaţie:
Un posibil *tipar* este prezentat în continuare:

```
class Vector {
  void **a;
  int capacity;
  int count;
public:
  Vector(Vector&);
  Vector(int _capacity);
  ~Vector();
  void addElement(void*); // adăugare element la început/final de vector
  void insertElement(void*, int poz); // inserare element în poziţia impusă
  void removeElement(void*); // elimină elementul de pe prima/ultima poziţie
  void removeAllElements(); // goleşte vectorul
  void* operator[] (int index); // operatorul de indexare; trebuie supraîncărcat
  friend ostream& operator<< (ostream& , Vector& ); // operatorul de afişare la stdout;
  // trebuie supraîncărcat.
};
```

(14) COADĂ CIRCULARĂ

Implementaţi o structură de tip *coadă circulară*.
Ţineţi cont de implementarea propusă şi scrieţi un scurt program, în care să verificaţi funcţionalitatea propusă.
Indicaţie:
O posibilă implementare ar putea ţine cont de următoarele aspecte:

```
class Queue{
```

```
    void** tab;
    int first, last;
    int capacity;
  public:
    Queue(); // constructor implicit
    Queue(int _capacity); // constructor cu parametru
    Queue(Queue& ); // constructor de copiere
    ~Queue();
    void* deQueue(); // şterge element din coadă
    void enQueue(void*); // adaugă element în coadă
    int isEmpty(); // test coadă vidă (fără elemente)
    int isFull(); // test coadă plină (maximum de elemente, sau lungimea maximă
    // a cozii; se va ţine cont de variabila membru 'capacity')
};
```

(15) Mesaje de tip E-mail [4]

Proiectaţi o clasă `Message` care modelează un mesaj *de tip e-mail*.

Cerinţe de proiectare:

1) Un astfel de mesaj va conţine:
   a) un destinatar (*recipient*),
   b) un expeditor (*sender*),
   c) textul mesajului.
2) Clasa va oferi următoarea *interfaţă*:
   i. Un constructor ce preia destinatarul, recipientul şi stabileşte un time stamp al mesajului la *timpul curent*.
   ii. O funcţie membră care adaugă o linie de text la corpul mesajului
   iii. O funcţie membră `to_string()` care converteşte mesajul într-o singură linie de text, similar cu:
   `"From: My_Hacker\nTo : Your_Hacker\n ..."`
   iv. O funcţie membră care afişează un mesaj de tip text, construit conform cerinţelor enunţate.
   Indicaţie: Folosiţi funcţia construită anterior, `to_string()`.
3) Scrieţi un program care foloseşte această clasă pentru a construi un mesaj şi a-l afişa.

Bibliografie selectivă

[1] Emanuela Cerchez, Marinel Şerban, *Programarea în limbajele C/C++ pentru liceu. Programare orientată pe obiecte şi programare generică, vol. 4,* Editura Polirom, Bucureşti, 2013

[2] Scott Meyers, *Effective C++, 3^{rd} edition,* Addison Wesley Professional, 2005, ISBN:

[3] Cay Horstmann, *C++ for everyone, 2^{nd} edition,* John Wiley & Sons, 2009-2012

[4] Mihai Gabroveanu, *Exerciţii şi probleme,* `http://inf.ucv.ro/$\sim$mihaiug/courses/poo/labs/Probleme%20C++.pdf`

[5] Mihai Gabroveanu, *Exerciţii şi probleme,* `http://inf.ucv.ro/~mihaiug/courses/poo/labs/Lista%20Probleme%20C++%20II.pdf`

[6] Steve Oualline, *Practical C++ Programming,* O'Reilly, 2002, ISBN:0-596-00419-2

[7] Jeff Cogswell, Christopher Diggins, Ryan Stephens, Jonathan Turkanis, *C++ cookbook,* O'Reilly, 2005, ISBN:0-596-00761-2

[8] Stanley B. Lippman, Josée Lajoie, Barbara E. Moo, *C++ Primer, 4^{th} Edition,* Addison Wesley Professional, 2005, ISBN:0-201-72148-1

[9] Peter J. Deitel, Harvey M. Deitel, *C++: how to program, 8^{th} edition* PearsonEducation,Inc. and Prentice Hall, 2001-2012, ISBN:978-0-13-266236-9