Exercițiul 4

1. (5p) Să se definească o clasă `TimePeriod2`. O instanță a clasei `TimePeriod2` păstrează o perioadă de timp exprimată *în ore şi minute*. Clasa `TimePeriod2` preia funcționalitatea clasei `TimePeriod` (*a se vedea Exercițiul 1 anterior*).

```
class TimePeriod2 {
    int h, m;
public:
    TimePeriod2(int ore, int minute);
    TimePeriod2(const TimePeriod2& tp);

    TimePeriod2& operator= (const TimePeriod2& tp);

    inline int getHours() const;
    inline int getMinutes() const;

    string getstring();

    TimePeriod2& operator+= (const TimePeriod2& tp);
    TimePeriod2 operator+(const TimePeriod2& tp);
    TimePeriod2& operator++(int);
    TimePeriod2& operator+(int);

    friend ostream& operator << (ostream& os, const TimePeriod2& tp);
};
```

Elementele noi din clasa `TimePeriod2` (față de clasa `TimePeriod`) sunt următoarele:

- `operator= (const TimePeriod2& tp)` – inițializează instanța curentă a clasei (`this`) cu valorile unei alte instanțe a clasei identificată de `tp`. Reprezintă operatorul '=' supraîncărcat.
- `operator+= (const TimePeriod2& tp)` – adaugă o perioadă de timp dată sub forma unui obiect de tip `TimePeriod2` la perioada de timp curentă (obiectul curent identificat de către `this`). Metoda este similară ca funcționalitate cu metoda `add(timeperiod)` din clasa `TimePeriod`. Întoarce o referință către obiectul curent (nu se creează un obiect nou). Reprezintă operatorul '+=' supraîncărcat.
- `operator+(const TimePeriod2& tp)` – adaugă o perioadă de timp dată sub forma unui obiect de tip `TimePeriod2` la perioada de timp existentă şi întoarce rezultatul sub forma unui obiect nou de acelaşi tip (exemplu: `tp2 = tp1 + tp;`). Reprezintă operatorul '+' supraîncărcat.

- `operator++(int)` – incrementează cu o unitate numărul de minute corespunzător perioadei de timp curente (exemplu: `tp++;`). Reprezintă operatorul de *postincrementare* '++' supraîncărcat. Întoarce o referință către obiectul curent (nu se creează un obiect nou).
- `operator+(int dt)` – incrementează cu valoarea `dt` numărul de minute corespunzător perioadei de timp curente (exemplu: `tp1 = tp + 10;`). Întoarce o referință către obiectul curent (nu se creează un obiect nou).
- `operator<< (ostream& os, const TimePeriod2& tp)` – trimite către fluxul de date de ieşire identificat de către `os` un şir de caractere reprezentând numărul de ore şi de minute corespunzător perioadei de timp curente, de forma "`Perioada: [hh ore - mm minute]`".

Referințe

clasa `TimePeriod` (*din Exercițiul 1*), clasa `Rational`, clasa `Complex`.

Codul de testare al clasei `TimePeriod2`:

```
int main(void) {
  TimePeriod2 tp1(1, 30);
  TimePeriod2 tp2 = tp1;

  cout << tp1;

  tp2 += tp1;
  cout << tp2;

  cout << "Perioada 2: " << tp2.getstring() << '\n';

  TimePeriod2 tp3(tp1.getHours(), tp2.getMinutes());
  cout << tp3;

  tp3 = tp1 + tp2;
  cout << tp3;
  cout << "Perioada 3: " << tp3.getstring() << '\n';

  tp3 = tp3 + 29;
  cout << tp3;

  tp3++;
  cout << tp3;

  return 0;
}
```

2. (5p) Să se implementeze o clasă `MultimeDeSiruri`, ce reprezintă o mulțime de şiruri de caractere (care nu se repetă).

```
#define MAX_SIZE 100

class MultimeDeSiruri {
  private:
    int count;
    string v[MAX_SIZE];

  public:
    MultimeDeSiruri();
    MultimeDeSiruri(int n, string a[]);
    MultimeDeSiruri(const MultimeDeSiruri& a);

    inline int size() const;
    int contains(string str) const;
    int add(string str);

    void print() const;

    MultimeDeSiruri& operator= (const MultimeDeSiruri& a);
    MultimeDeSiruri operator+(const MultimeDeSiruri& a);
    MultimeDeSiruri operator* (const MultimeDeSiruri& a);

    friend ostream& operator<< (ostream& os, const MultimeDeSiruri& a);
};
```

Clasa `MultimeDeSiruri` va conține următoarele elemente (a se vedea declarația clasei):

- două atribute private, `count` – numărul de elemente din mulțime la un moment dat, şi `v` – un tablou în care sunt păstrate valorile mulțimii, şiruri de caractere (obiecte de tip `string`).
- un constructor fără parametri `MultimeDeSiruri()`, ce inițializează obiectul cu mulțimea vidă.
- un constructor cu parametri `MultimeDeSiruri(int n, string ts[])`, ce inițializează mulțimea cu valorile din tabloul `ts` trimis ca parametru. `n` păstrează numărul de elemente din tabloul `ts`. Se verifică ca elementele din tabloul `ts` să nu fie duplicate.
- un constructor de copiere `MultimeDeSiruri(const MultimeDeSiruri &)`.
- o metodă inline `size()` ce întoarce numărul de elemente al multimii.
- o metodă `contains(string)` pentru a verifica dacă un şir de caractere aparține mulțimii curente (întoarce valoarea `1` în caz afirmativ şi `0` în cazul în care nu aparține).

- o metodă `add(string)` ce adaugă şirul de caractere transmis ca parametru la mulțimea curentă, dacă acesta nu există deja (întoarce valoarea `1` în cazul în care adăugarea se efectuează şi `0` în caz contrar).
- o metodă `print()` ce afişeaza elementele mulțimii curente.
- `operator= (const MultimeDeSiruri& ma)` – operatorul de copiere. `ma` este o referință către o mulțime de şiruri de caractere ce se va copia în obiectul curent.
- `operator+(const MultimeDeSiruri& ma)` - calculează reuniunea dintre două mulțimi de şiruri de caractere: primul operand este determinat de `this` iar cel de-al doilea operand este determinat de `ma`. Întoarce rezultatul operației într-un obiect nou creat.
- `operator* (const MultimeDeSiruri& ma)` - calculează intersecția dintre două mulțimi de şiruri de caractere: primul operand este determinat de `this` iar cel de-al doilea operand este determinat de `ma`. Întoarce rezultatul operației într-un obiect nou creat.
- `operator<< (ostream& os, const MultimeDeSiruri& ma)` - va afişa elementele mulțimii `ma`. "Afişarea" se realizează către fluxul de date de ieşire; "`Multime #size [element1, element2, ...]`".

Referințe

Clasa `Circle`, clasa `Rational`, clasa `Complex`.

Codul de testare al clasei `MultimeDeSiruri`:

```
int main() {
  MultimeDeSiruri a;
  string flag1[] = {"alb", "rosu", "verde", "galben", "rosu"};
  int i;

  for (i = 0; i < sizeof(flag1) / sizeof(flag1[0]); i++) {
    a.add(flag1[i]);
  }
  cout << a << '\n';

  MultimeDeSiruri b;
  string flag2[] = {"rosu", "galben", "albastru", "galben", "albastru"};

  for (i = 0; i < sizeof(flag2) / sizeof(flag2[0]); i++) {
    b.add(flag2[i]);
  }
  b.print();

  MultimeDeSiruri c = a + b;
  cout << c << '\n';

  c = a * b;
  c.print();

  return 0;
}
```