Esercizio n. 1

In una applicazione in cui si richiede di gestire i dati relativi ad una banca facendo uso delle classi: Conto e Servizio. La prima classe consente di memorizzare i dati relativi ad un conto corrente ed implementa un array dinamico di oggetti di tipo Servizio. Seguono le intestazioni delle classi:

```
class Servizio
{
friend ostreamOperator<<(ostream&, const Servizio&);
private:
    MyString codice;    MyString nome;    double costo;
public:
    Servizio(MyString c, MyString n, double c);
    Servizio(const Servizio&);
    bool operator==(const Servizio&);
    MyString getCodice();
    MyString getNome();
    double getCosto();
    void setCodice(MyString);
    void setNome(MyString);
    void setCosto(Double);
};

class Conto
{
friend ostreamOperator<<(ostream&, const Conto &);
private:
    int numero;
    MyString Intestatario;    Servizio* servizi;
    double saldo;  int lunghezza;
public:
    Conto (int = 0, MyString i, double = 0, int = 0);
    Conto (const Conto&);
    Servizio& operator[](int);
    int numeroServizi();
    int canoneMensile();
    int getNumero();
    MyString getIntestatario();
    double getSaldo();
    void versa(double);
    void setNumero(int);
    ~Conto();
};
```

Si implementino **solo** i metodi di seguito elencati, supponendo già implementati i rimanenti.

- Costruttore della classe Servizio
- Metodo getCosto() della classe Servizio
- Operatore << della classe Servizio
- Costruttore di copia della classe Conto
- Metodo canoneMensile() della classe Conto
- Distruttore classe Conto

Il metodo canoneMensile della classe Conto restituisce i costi di gestione di un conto corrente che sono pari alla somma dei costi di tutti i servizi associati al conto in questione più una quota fissa di 2 euro.

Nota: E' possibile aggiungere, qualora lo studente lo ritenga necessario, ulteriori membri pubblici o privati alle classi. Nel caso in cui venga aggiunta una nuova funzione deve esserne fornita l'implementazione.

Esercizio 2

Siano date le classi Conto (definita nell'esercizio 1) e Filiale, che rappresentano rispettivamente i dati relativi ai conti correnti e alle filiali di una banca. Sia l'intestazione della classe Filiale la seguente:

```
class Filiale
{
friend ostreamOperator<<(ostream&, const Produttore &);
private:
    MyString nome;
    ListaConto conti_corrente;
public:
    Filiale (MyString n);
    Filiale (const Filiale &);
    bool operator==(const Produttore &);
    void inserisciConto(Conto);
    MyString getNome();
};
```

Si assume che i codici dei conto correnti e siano univoci; inoltre, si assume essere data l'implementazione delle classi Conto e Filiale oltre a quella delle classi ListaConto e IteratoreConto (che sono definite come di consueto).

Utilizzando gli iteratori scrivere le funzioni:

- `ListaConto  contiInRosso(const Filiale& f);`
- `bool clientiComuni(const Filiale& f1, const Filiale& f2);`

La prima funzione, data una filiale determina la lista dei conti correnti con saldo negativo. Mentre la seconda funzione, date due filiali restituisce *true* se esse hanno un cliente in comune, *false* altrimenti.

Nota: La funzione `clientiComuni` restituisce *true* se uno stesso cliente ha (almeno) un conto in entrambe le filiali specificate in input.

Esercizio 3

Data la classe *Conto* descritta nell'esercizio 1, si utilizzi l'ereditarietà per progettarne una estensione dal nome *ContoStudente*. Un particolare conto corrente per studenti differisce da quello classico per quanto riguarda il calcolo del costo mensile, che è sempre pari a zero.
È richiesta l'implementazione dell'interfaccia della classe ContoStudente e del metodo canoneMensile().