FEJLETT PROGRAMOZÁSI TECHNIKÁK (C++)
7. GYAKORLAT

Cél:

- Statikus adattagok
  - ID generálás
- Operátorok túlterhelése
  - inserter operátor (operator<<)
  - index operátor
- Osztályok közötti kapcsolatok
  - tartalmazási kapcsolat
  - egy-a-sokhoz kapcsolat megvalósítása

1. Feladat - Account osztály

Adott egy Account.h állomány, amely egy bankszámla osztályt deklarál. Az osztály tegye lehetővé az id mező automatikus előállítását a statikus counter mező használatával.

```
#ifndef ACCOUNT_H
#define ACCOUNT_H

#include <iostream>
using namespace std;

class Account {
public:
   Account(double balance=0);
   void deposit( double amount);
   bool withdraw(double amount);
   int getId() const;
   double getBalance() const;
   void print(ostream& os) const;
   friend ostream& operator<<(ostream& os, const Account& account);
private:
   static int counter;
   int id; //generated field
   double balance;
};
#endif
```

- Implementálja az Account osztályt. Figyeljen a statikus adattagok helyes inicializálására.
- Tesztelje az osztályt úgy, hogy hozzon létre egy Account példányt 0 egyenleggel. Tegyen be a számlára 1000 RON-t, majd vegyen ki először 500 RON-t, utána pedig 1000 RON-t. Minden művelet után írassa ki a számla állapotát (id és balance mezők értékei).

2. Feladat - Customer osztály

Adott egy Customer.h állomány, amely egy banki ügyfelet deklarál.

- Az id mező generálása az Account osztályhoz hasonlóan történik a counter statikus adattag segítségével.
- Az ügyfél számláit az accounts dinamikus tömbben tároljuk.
- A getAccount(id) függvény visszatéríti a megadott azonosítójú számlát.

- Az index operátor a megadott *sorszámú* bankszámlát téríti vissza.

```
#ifndef CUSTOMER_H
#define CUSTOMER_H
#include <string>
#include "Account.h"
#include <vector>
using namespace std;

class Customer {
private:
   int id; //generated field
   string firstName;
   string lastName;
   vector<Account> accounts;
   static int counter;
public:
   Customer(const string& firstName, const string& lastName);
   const string &getFirstName() const;
   void setFirstName(const string &firstName);
   const string &getLastName() const;
   void setLastName(const string &lastName);
   int getId() const{ return id;}
   Account& getAccount(int id);
   //returns the ID of the created account
   int newAccount(double balance);
   bool deleteAccount(int id);
   int getNumAccounts() const;
   void print(ostream& os) const;
   friend ostream& operator<<(ostream& os, const Customer& customer);
   Account& operator[](int index);
   const Account& operator[](int index) const;
};
#endif
```

- Implementálja a Customer osztályt.
- Hozzon létre egy Customer példányt.
- Nyisson két bankszámlát a megadott ügyfélnek, az elsőnek legyen 0 kezdőegyenlege, a másodiknak pedig 1000 RON.
- Írassa ki az ügyfelet az inserter operátor segítségével. A kiíratás legyen szépen formázott.

```
1 Biro Tamas
      Account Id: 1 balance: 0
      Account Id: 2 balance: 1000
```

3. Feladat - Bank osztály

```
#ifndef BANK_H
#define BANK_H
#include <string>
#include "Customer.h"
using namespace std;

class Bank {
public:
   Bank(const string&);
   //returns the ID of the new Customer
   int newCustomer(const string& firstName, const string& lastName );
   bool deleteCustomer(int id);
   Customer& getCustomer(int id);

   //Convenience functions
   void printCustomers(ostream& os=cout) const;
   void printCustomersAndAccounts(ostream& os=cout) const;
   //LOAD customers, returns their IDs
   vector<int> loadCustomers(const string& filename);
private:
   vector<Customer> customers;
   string name;
};
#endif
```

- Hozzon létre egy bankot.
- Olvassa be a bank ügyfeleit egy customers.txt állományból (minden sor egy ügyfél vezeték és keresztnevét tartalmazza, a fájl legkevesebb 3 ügyfelet tartalmazzon).
- Írassa ki a bank ügyfeleit.
- Minden beolvasott ügyfélhez rendeljen hozzá legalább két számlát, majd végezzen műveleteket a számlákkal. Minden műveletet a bank példányon keresztül kell végezni!
- A műveletek után írassa ki az ügyfeleket a számlákkal együtt!