Rejstřík firem II

Termín odevzdání:	**31.03.2013 23:59:59**
Pozdní odevzdání s penalizací:	**12.05.2013 23:59:59** (Penále za pozdní odevzdání: 100.0000 %)
Hodnocení:	**4.4000**
Max. hodnocení:	**4.0000** (bez bonusů)
Odevzdaná řešení:	1 / 20 Volné pokusy + 20 Penalizované pokusy (-2 % penalizace za každé odevzdání)
Nápovědy:	0 / 2 Volné nápovědy + 2 Penalizované nápovědy (-10 % penalizace za každou nápovědu)

Tato úloha je pokračováním jednodušší verze firemního rejstříku. Doporučujeme nejprve vypracovat úlohu snazší a teprve potom začít pracovat na této úloze.

Zadání úlohy je prakticky totožné. Rozdílem je, že v této úloze jeden vlastník může vlastnit více firem a firma může být vlastněna více vlastníky. Opět předpokládáme identifikaci pomocí jména a adresy, opět předpokládáme, že jméno a adresa vlastníka musí být mezi vlastníky unikátní a to samé platí pro jméno a adresu firmy mezi firmami.

Provedená úprava se projeví na rozhraní požadovaných tříd. Změna nastane hlavně u metod vyhledávacích, které budou muset vracet seznam nalezených vlastníků / firem. Požadované rozhraní je nyní tvořeno dvěma třídami:

```
#ifndef __PROGTEST__
#include <iostream>
#include <iomanip>
#include <string>
#include <cstring>
#include <cstdlib>
#include <cstdio>
using namespace std;
#endif /* __PROGTEST__ */

class CIterator
 {
   public:
    bool           AtEnd          ( void ) const;
    void           Next           ( void );
    const string & Name           ( void ) const;
    const string & Address        ( void ) const;
    // todo
 };

class CCompanyIndex
 {
   public:
                 CCompanyIndex    ( void );
                ~CCompanyIndex    ( void );
    bool         Add              ( const string & oName,
                                    const string & oAddr,
                                    const string & cName,
                                    const string & cAddr );
    bool         Del              ( const string & oName,
                                    const string & oAddr,
                                    const string & cName,
                                    const string & cAddr );

    CIterator  * SearchOwner      ( const string & oName,
                                    const string & oAddr ) const;

    CIterator  * SearchCompany    ( const string & cName,
                                    const string & cAddr ) const;
    // todo
 };
```

Třída `CCompanyIndex` reprezentuje firemní rejstřík. Její veřejné rozhraní obsahuje:

- Konstruktor bez parametrů. Tento konstruktor inicializuje instanci třídy tak, že vzniklá instance je zatím prázdná (neobsahuje žádné záznamy).
- Destruktor. Uvolňuje prostředky, které instance alokovala.
- Metoda `Add(oName, oAddr, cName, cAddr)` přidá do existující databáze další záznam. Metoda vrací hodnotu `true`, pokud byl do záznam přidán, nebo hodnotu `false`, pokud přidán nebyl (protože již v databázi existoval záznam o tom, že daný majitel vlastní podíl v zadané firmě.). Parametry `oName` a `oAddr` reprezentují jméno a adresu vlastníka, parametry `cName` a `cAddr` udávají název a adresu firmy.
- Metoda `Del (oName, oAddr, cName, cAddr)` odstraní záznam z databáze. Pokud byl záznam skutečně odstraněn, vrátí metoda hodnotu `true`. Pokud záznam neodstraní (protože zadaný majitel neměl v zadané firmě podíl), vrátí metoda hodnotu `false`.
- Metoda `SearchOwner (oName, oAddr)` zjistí seznam všech firem, ve kterých má zadaný majitel podíl. Pokud zadaný majitel nemá podíl v žádné firmě, metoda vrátí návratovou hodnotu `NULL`. Pokud zadaný majitel má nějaký podíl v nějaké firmě, metoda vrátí dynamicky alokovanou instanci třídy `CIterator`, která obsahuje seznam všech firem, kde má zadaný majitel nějaký podíl. Po zpracování výsledku volající uvolní předanou instanci `CIterator` voláním `delete`.
- Metoda `SearchCompany` je obdoba volání `SearchOwner`. Metoda vyhledá a vrátí seznam podílníků zadané firmy, případně `NULL` pokud je seznam podílníků prázdný.

Třída `CIterator` reprezentuje seznam nalezených firem nebo podílníků. Její veřejné rozhraní obsahuje:

- Metodu `AtEnd`. Tato metoda vrací `true` pokdu jsme již dosáhli konce seznamu, `false` pokud ještě v seznamu existují nějaké záznamy.
- Metoda `Next`. Tato metoda přejde v seznamu na další záznam. Průchod seznamem je pouze jednosměrný, není požadováno, aby bylo možné se vracet. Průchod seznamem volá v cyklu metodu `Next`, dokud nedosáhne konce seznamu - `AtEnd` vrátí `true`. Viz též ukázku dole.
- Metody `Name` a `Address` vrátí jméno a adresu uložené v aktuálním záznamu.
- Pořadí vracených jmen a adres není důležité, je však nutné vrátit postupně všechny relevantní dvojice.

Odevzdávejte soubor, který obsahuje implementované třídy `CCompanyIndex` a `CIterator`. Třídy musí splňovat veřejné rozhraní podle ukázky - pokud Vámi odevzdané řešení nebude obsahovat popsané rozhraní, dojde k chybě při kompilaci. Do tříd si ale můžete doplnit další metody (veřejné nebo i privátní) a členské proměnné. Odevzdávaný soubor musí obsahovat jak deklaraci tříd (popis rozhraní) tak i definice metod, konstruktoru a destruktoru. Je jedno, zda jsou metody implementované inline nebo odděleně. Odevzdávaný soubor nesmí kromě implementace tříd obsahovat nic jiného, zejména ne vkládání hlavičkových souborů a funkci `main` (funkce `main` a vkládání hlavičkových souborů může zůstat, ale pouze obalené direktivami podmíněného překladu jako v ukázce níže).

Implementované třídy nesmí používat datové struktury z STL (vector, list, ...). Jejich použití povede k chybě při kompilaci. Třídy jsou testované v omezeném prostředí, kde je limitovaná dostupná paměť (dostačuje k uložení seznamu) a je omezena dobou běhu.

Implementované třídy se nemusí zabývat kopírujícím konstruktorem ani přetěžováním operátoru =. V této úloze ProgTest neprovádí testy této funkčnosti.

Implementace tříd musí být efektivní z hlediska nároků na čas i nároků na paměť. Jednoduché lineární řešení nestačí (pro testovací data vyžaduje čas přes 10 minut, limit je 10s). Předpokládejte, že operace dotazu je cca 10x častější než vkládání a mazání. Proto vyhovuje řešení s polem seřazených hodnot, kde vyhledávání probíhá půlením intervalu s logaritmickou složitostí. Méně časté operace vkládání a výmaz je pak doporučeno implementovat postupem, který v logaritmickém čase najde pozici vkládaného/odebíraného prvku a pak v lineárním čase posune obsah pole.

Pro uložení hodnot alokujte pole dynamicky. Počáteční velikost volte malou (např. tisíc prvků) a velikost zvětšujte/zmenšujte podle potřeby. Při zaplnění pole není vhodné alokovat nové pole větší pouze o jednu hodnotu, takový postup má obrovskou režii na kopírování obsahu. Je rozumné pole rozšiřovat s krokem řádově tisíců prvků, nebo geometrickou řadou s kvocientem ~1.5 až 2.

Ukázka použití třídy:

```
void  showResults ( CIterator * it )
 {
   while ( ! it -> AtEnd () )
    {
      cout << it -> Name () << ",  " << it -> Address () << endl;
      it -> Next ();
    }
 }

...

CIterator * it;
bool        status;
CCompanyIndex  b1;
```

```
status = b1 . Add ( "Smith", "Oak road", "ACME, Ltd.", "One ACME road" );
// status = true
status = b1 . Add ( "Brown", "Second street", "ACME, Ltd.", "Mountain road" );
// status = true
status = b1 . Add ( "Hacker", "5-th avenue", "Forks and Knives, Ltd.", "Cutlery avenue" );
// status = true
status = b1 . Add ( "Hacker", "7-th avenue", "Child toys, Inc.", "Red light district" );
// status = true
status = b1 . Add ( "Smith", "Oak road", "ACME, Ltd.", "Mountain road" );
// status = true
status = b1 . Add ( "Hacker", "5-th avenue", "ACME, Ltd.", "One ACME road" );
// status = true
status = b1 . Add ( "Hacker", "7-th avenue", "ACME, Ltd.", "Mountain road" );
// status = true
it = b1 . SearchOwner ( "Brown", "Second street" );
showResults ( it );
/*
----8<----8<----8<----8<----
ACME, Ltd., Mountain road
----8<----8<----8<----8<----
*/
delete it;
it = b1 . SearchOwner ( "Hacker", "Oak road" );
// it = NULL
it = b1 . SearchOwner ( "Hacker", "7-th avenue" );
showResults ( it );
/*
----8<----8<----8<----8<----
Child toys, Inc., Red light district
ACME, Ltd., Mountain road
----8<----8<----8<----8<----
*/
delete it;
it = b1 . SearchCompany ( "ACME, Ltd.", "Mountain road" );
showResults ( it );
/*
----8<----8<----8<----8<----
Hacker, 7-th avenue
Brown, Second street
Smith, Oak road
----8<----8<----8<----8<----
*/
delete it;
it = b1 . SearchCompany ( "Child toys, Inc.", "Mountain road" );
// it = NULL
status = b1 . Del ( "Smith", "Oak road", "Child toys, Inc.", "Red light district" );
// status = false
status = b1 . Del ( "Smith", "Oak road", "ACME, Ltd.", "Mountain road" );
// status = true
it = b1 . SearchOwner ( "Smith", "Oak road" );
showResults ( it );
/*
----8<----8<----8<----8<----
ACME, Ltd., One ACME road
----8<----8<----8<----8<----
*/
delete it;
status = b1 . Add ( "Smith", "Oak road", "ACME, Ltd.", "One ACME road" );
// status = false
```

Referenční řešení

1	**20.03.2013 15:38:42**	**Download**
Stav odevzdání:	Ohodnoceno	
Hodnocení:	4.4000	

- **Hodnotitel: automat**
  - Program zkompilován
  - Test 'Zakladni test s parametry podle ukazky': Úspěch
    - Dosaženo: 100.00 %, požadováno: 100.00 %
    - Celková doba běhu: 0.000 s (limit: 2.000 s)
    - Využití paměti: 12576 KiB (limit: 23126 KiB)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test nahodnymi vstupy (Add, Search)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.225 s (limit: 2.000 s)
    - Využití paměti: 19200 KiB (limit: 23126 KiB)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test nahodnymi vstupy (Add, Del, Search)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.223 s (limit: 1.775 s)
    - Využití paměti: 19268 KiB (limit: 23126 KiB)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test nahodnymi hodnotami + test prace s pameti': Úspěch
    - Dosaženo: 100.00 %, požadováno: 25.00 %
    - Celková doba běhu: 4.076 s (limit: 10.000 s)
    - Využití paměti: 26484 KiB (limit: 41085 KiB)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test rychlosti (nahodne hodnoty)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 1.233 s (limit: 10.000 s)
    - Využití paměti: 17320 KiB (limit: 32892 KiB)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test rychlosti (nahodna jmena, fixni adresa)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 75.00 %
    - Celková doba běhu: 1.213 s (limit: 8.767 s)
    - Využití paměti: 17320 KiB (limit: 32892 KiB)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Test 'Test rychlosti (fixni jmena, nahodna adresa)': Úspěch
    - Dosaženo: 100.00 %, požadováno: 75.00 %
    - Celková doba běhu: 1.333 s (limit: 7.554 s)
    - Využití paměti: 17320 KiB (limit: 32892 KiB)
    - Úspěch v nepovinném testu, hodnocení: 100.00 %
  - Všechny paměťové bloky byly uvolněné - ok.
  - Celkové hodnocení: 100.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.00 * 1.00)
- Celkové procentní hodnocení: 100.00 %
- Bonus za včasné odevzdání: 0.40
- Celkem bodů: 1.00 * (4.00 + 0.40) = 4.40

SW metriky:

	Celkem	Průměr	Maximum	Jméno funkce
Funkce:	**33**	--	--	--
Řádek kódu:	**319**	**9.67 ± 9.84**	**47**	`main()`
Cyklomatická složitost:	**72**	**2.18 ± 3.07**	**13**	`findElm(const string &,const string &,int,int,int &,CRecord **,int)`