Evidence zaměstnanců

Termín odevzdání:	**01.04.2018 23:59:59**
Pozdní odevzdání s penalizací:	**01.07.2018 23:59:59** (Penále za pozdní odevzdání: 100.0000 %)
Hodnocení:	**5.5000**
Max. hodnocení:	**5.0000** (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)

Úkolem je realizovat třídu `CPersonalAgenda`, která bude implementovat databázi zaměstnanců.

Pro usnadnění práce HR oddělení máme realizovanou databázi zaměstnanců. Pro sledované zaměstnance si pamatujeme jméno, příjmení, email a plat. V průběhu evidence zaměstnance se tyto údaje mohou měnit, údaje chceme vyhledávat a chceme mít přehled o platech.

Zaměstnanec je identifikován svým jménem, a příjmením nebo svým e-mailem. Firemní postupy zajišťují, že e-mail je unikátní přes celou databázi. Dále, jména a příjmení se mohou opakovat, ale dvojice (jméno, příjmení) je opět v databázi unikátní. Tedy v databázi může být mnoho lidí s příjmením `Svoboda`, mnoho lidí může mít jméno `Petr`, ale zaměstnanec `Svoboda Petr` může být v databázi v daném okamžiku pouze jeden. Při porovnávání jmen, příjmení i e-mailů rozlišujeme malá a velká písmena (case sensitive).

Veřejné rozhraní je uvedeno níže. Obsahuje následující:

- 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(name, surname, email, salary)` přidá do existující databáze další záznam. Parametry `name` a `surname` reprezentují jméno a příjmení, parametr `email` udává mailovou adresu a `salary` plat. Metoda vrací hodnotu `true`, pokud byl záznam přidán, nebo hodnotu `false`, pokud přidán nebyl (protože již v databázi existoval záznam se stejným jménem a příjmením, nebo záznam se stejným e-mailem).
- Metody `Del (name, surname) / Del (email)` odstraní záznam z databáze. Parametrem je jednoznačná identifikace pomocí jména a příjmení (první varianta) nebo pomocí emailu (druhá varianta). Pokud byl záznam skutečně odstraněn, vrátí metoda hodnotu `true`. Pokud záznam odstraněn nebyl (protože neexistoval zaměstnanec s touto identifikací), vrátí metoda hodnotu `false`.
- Metoda `ChangeName(email, newName, newSurname)` změní zaměstnanci jeho jméno a příjmení. Zaměstnanec je identifikován pomocí e-mailu `email`, jméno zaměstnance je změněno na `newName/newSurname`. Metoda vrátí `true` pro úspěch, `false` pro chybu (neexistuje zaměstnanec s takovým e-mailem, nově přidělené jméno/příjmení není unikátní).
- Metoda `ChangeEmail(name, surname, newEmail)` změní zaměstnanci jeho email. Zaměstnanec je identifikován pomocí jména a příjmení `name/surname`, e-mail zaměstnance je změněn na `newEmail`. Metoda vrátí `true` pro úspěch, `false` pro chybu (neexistuje zaměstnanec s takovým jménem a příjmením, nově přidělený e-mail není unikátní).
- Metody `SetSalary (name, surname, salary) / SetSalary (email, salary)` změní výši výplaty zadanému zaměstnanci. Varianty jsou dvě - zaměstnanec je identifikován buď svým jménem a příjmením, nebo svojí e-mailovou adresou. Pokud metoda uspěje, vrací `true`, pro neúspěch vrací `false` (neexistující zaměstnanec).
- Metody `GetSalary (name, surname) / GetSalary (email)` zjistí výši výplaty zadaného zaměstnance. Varianty jsou dvě - zaměstnanec je identifikován buď svým jménem a příjmením, nebo svojí e-mailovou adresou. Pokud metoda uspěje, vrací výši výplaty zaměstnance, pro neúspěch (neexistující zaměstnanec) vrací hodnotu 0.
- Metody `GetRank (name, surname, rankMin, rankMax) / GetRank (email, rankMin, rankMax)` zjistí jak dobře je zaměstnanec placen ve vztahu k ostatním. Výsledkem je pozice výplaty zadaného zaměstnance na pomyslném žebříčku výplat od nejhorší (nejnižší) k nejlepší (nejvyšší). Parametrem je identifikace zaměstnance (podle varianty buď jménem a příjmením, nebo e-mailovou adresou), parametry `rankMin/rankMax` jsou výstupní, do nich funkce zapíše pozici výplaty hledaného zaměstnance v žebříčku. Protože stejnou výplatu může dostávat více zaměstnanců, je výstupem dvojice hodnot - interval min-max. Pokud například hledaný zaměstnanec dostává výplatu 20000, 37 zaměstnanců dostává výplatu nižší a 21 dalších zaměstnanců dostává výplatu stejnou (tedy plat 20000 dostává celkem 22 zaměstnanců), pak výsledkem je `rankMin=37` a `rankMax=37+22-1=58`. Návratovou hodnotou funkce je `true` pro úspěch (zadaný zaměstnanec nalezen, výstupní parametry vyplněny) nebo `false` (zadaný zaměstnanec nenalezen, výstupní parametry ponechány beze změn).
- Metoda `GetFirst ( outName, outSurname )` slouží pro procházení databází. Zaměstnance chceme procházet podle abecedy (seřazené vzestupně podle příjmení, v případě stejných příjmení podle jména). Metoda vrátí prvního zaměstnance v takto seřazeném seznamu, jeho jméno a příjmení zapíše do zadaných výstupních parametrů `outName/outSurname`. Návratovou hodnotou je `true` pro úspěch (databáze nebyla prázdná) nebo `false` (prázdná databáze, výstupní parametry ponechány beze změn).
- Metoda `GetNext ( name, surname, outName, outSurname )` slouží k procházení zaměstnanců podobně jako metoda `GetFirst`. Tato metoda vrátí dalšího zaměstnance, který v seřazeném seznamu zaměstnanců (viz `GetFirst`) následuje po zaměstnanci `name/surname`. Jméno následujícího zaměstnance zapíše do zadaných výstupních parametrů `outName/outSurname`. Návratovou hodnotou je `true` pro úspěch (zaměstnanec `name/surname` nalezen a

není poslední v seřazeném seznamu) nebo `false` pro neúspěch (zaměstnanec `name/surname` nenalezen nebo je poslední v seznamu). V případě neúspěchu metoda nebude měnit výstupní parametry `outName/outSurname`.

Odevzdávejte soubor, který obsahuje implementovanou třídu `CPersonalAgenda`. Třída musí splňovat veřejné rozhraní podle ukázky - pokud Vámi odevzdané řešení nebude obsahovat popsané rozhraní, dojde k chybě při kompilaci. Do třídy si ale můžete doplnit další metody (veřejné nebo i privátní) a členské proměnné. Dále si do odevzdávaného souboru můžete doplnit další podpůrné funkce nebo třídy. Odevzdávaný soubor musí obsahovat jak deklaraci třídy (popis rozhraní), tak i definice metod, konstruktoru a destruktoru. Je jedno, zda jsou metody implementované inline nebo odděleně. Odevzdávaný soubor nesmí obsahovat vkládání hlavičkových souborů a funkci `main` (funkce `main` a vkládání hlavičkových souborů může zůstat, ale pouze obalené direktivami podmíněného překladu). Za základ implementace použijte přiložený zdrojový soubor.

Rozhraní třídy obsahuje řadu metod ve dvou variantách, které se liší pouze způsobem identifikace zaměstnance. Je vhodné věnovat nenulový čas návrhu třídy tak, abyste všechen výkonný kód nekopírovali 2x (např. realizujte privátní metody, které budete volat z více veřejných metod).

Třída je testovaná v omezeném prostředí, kde je limitovaná dostupná paměť (dostačuje k uložení seznamu) a je omezena dobou běhu. Implementovaná třída se nemusí zabývat kopírujícím konstruktorem ani přetěžováním operátoru =. V této úloze ProgTest neprovádí testy této funkčnosti.

Implementace třídy musí být efektivní z hlediska nároků na čas i nároků na paměť. Jednoduché lineární řešení nestačí (pro testovací data vyžaduje čas přes 5 minut). Předpokládejte, že vkládání a mazání zaměstnance jsou řádově méně časté než ostatní operace, tedy zde je lineární složitost akceptovatelná. Častá jsou volání `GetSalary` a `SetSalary`, jejich časová složitost musí být lepší než lineární (např. logaritmická nebo amortizovaná konstantní). Dále, metody `GetFirst/GetNext` by též měly být efektivní, jejich složitost by též měla být lepší než lineární.

V povinných testech se metoda `GetRank` volá málo často, tedy nemusí být příliš efektivní (pro úspěch v povinných testech stačí složitost lineární nebo `n log n`, pro bonusový test je potřeba složitost lepší než lineární. Pokud nechcete vymýšlet efektivní algoritmus pro bonusový test, zaměřte se spíše na to, aby volání `SetSalary` byla efektivní i za cenu méně efektivní metody `GetRank`.

Bonusový test lze vyřešit několika způsoby. Při návrhu řešení můžete využít znalosti, že zadávaná hodnota mzdy je nejvýše 1000000. Dále může pomoci znalost, že hodnoty mezd se často opakují.

Pro uložení hodnot alokujte pole dynamicky případně použijte STL. Pozor, pokud budete pole alokovat ve vlastní režii, zvolte počáteční velikost 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.

Pokud budete používat STL, nemusíte se starat o problémy s alokací. Pozor - k dispozici máte pouze část STL (viz hlavičkové soubory v přiložené ukázce). Tedy například kontejnery `map` / `unordered_map` / `set` / `unordered_set` / ... nejsou k dispozici.

V přiloženém zdrojovém kódu jsou obsažené základní testy. Tyto testy zdaleka nepokrývají všechny situace, pro odladění třídy je budete muset rozšířit. Upozorňujeme, že testy obsažené v odevzdaných zdrojových kódech považujeme za nedílnou součást Vašeho řešení. Pokud v odevzdaném řešení necháte cizí testy, může být práce vyhodnocena jako opsaná.