Datum

Termín odevzdání:	**08.04.2018 23:59:59**
Pozdní odevzdání s penalizací:	**01.07.2018 23:59:59** (Penále za pozdní odevzdání: 100.0000 %)
Hodnocení:	**5.1839**
Max. hodnocení:	**5.0000** (bez bonusů)
Odevzdaná řešení:	10 / 25 Volné pokusy + 20 Penalizované pokusy (-2 % penalizace za každé odevzdání)
Nápovědy:	2 / 2 Volné nápovědy + 2 Penalizované nápovědy (-10 % penalizace za každou nápovědu)

Úkolem je realizovat třídu `CDate`, která bude reprezentovat datum.

Třída `CDate` usnadní práci s datem. Ve vhodné vnitřní reprezentaci si uloží zadaný rok, měsíc a den. Pro uložená data garantuje, že jsou platná, navíc nabízí následující operace:

konstruktor (y,m,d)
: inicializuje objekt, který bude reprezentovat zadané datum. Pokud datum není platné, konstruktor vyhodí výjimku `InvalidDateException`.

destruktor
: bude implementován, pokud to vnitřní struktury reprezentace vyžaduje.

kopírující konstruktor a přetížený operátor =
: bude umožňovat kopírování instancí. Implementujte jej pouze pokud automaticky generované varianty pro Vaši reprezentaci nevyhovují.

operátor <<
: bude umožňovat výstup do C++ streamu. Výpis bude realizován v podobě `YYYY-MM-DD`, kde den a měsíc jsou dvojmístné (případně zleva doplněné nulou), rok je zobrazen jako celé číslo (rok nemá žádné nároky na formátování, platné jsou pouze roky větší rovné 1600, viz níže).

operátory + a -
: umožní k datu přičíst nebo od data odečíst zadaný počet dní/měsíců/roků a tím vytvořit nové datum posunuté do budoucnosti nebo minulosti. Během operace se musí kontrolovat, zda nevzniklo neplatné datum. Pokud neplatné datum vznikne, operátor musí vyhodit výjimku `InvalidDateException`. Při sčítání/odečítání použijeme následující pravidla:

  - přičtení/odečtení roků změní odpovídajícím způsobem letopočet. Výjimka bude vyhozena, pokud se výsledné datum změní na neplatné (například z platného data 29.2.2000 se přičtením jednoho roku může stát neplatné datum 29.2.2001) nebo pokud se výsledný rok sníží pod hodnotu 1600.
  - přičítání/odečítání měsíců posune datum o zadaný počet měsíců zpět/vpřed. Lze přičítat/odečítat i velké množství měsíců, tím může dojít ke změně roku. Manipulací s měsíci se rovněž může změnit platné datum na neplatné (např. 31.3.2018 + 1 měsíc dává neplatné datum 31.4.2018), případně může dojít k chybě pokud se výsledný rok sníží pod hodnotu 1600. Obě popsané situace povedou k vyhození výjimky.
  - přičítání/odečítání dnů posune datum o zadaný počet dní, tím se může změnit i měsíc, případně i rok. Operace přičítání dnů nemůže vytvořit neplatné datum (např. 30.4.2018 + 2 dny dá platné datum 2.5.2018), k chybě ale může dojít překročením dolního limitu na rok (výsledný rok je menší než 1600). Tato situace povede k vyhození výjimky.

operátor -
: další varianta tohoto operátoru umožní odečíst dvě data, výsledkem je počet dní, které mezi nimi uplynuly,

operátory ==, != a <
: umožní porovnat data obvyklým způsobem,

operátor +=
: umožní změnit instanci data na levé straně tím, že ji posune v čase o zadaný počet dní/měsíců/let. Při úpravách se použijí stejná pravidla jako v případě operátoru +.

Odevzdávejte zdrojový soubor, který obsahuje Vaší implementaci třídy `CDate`. V odevzdávaném souboru nenechávejte vkládání hlavičkových souborů, Vaše testovací funkce a funkci `main`. Pokud v souboru chcete ponechat `main` nebo vkládání hlavičkových souborů, vložte je do bloku podmíněného překladu.

V tomto příkladu není poskytnutý přesný předpis pro požadované rozhraní třídy. Z textového popisu, ukázky použití v příloze a ze znalostí o přetěžování operátorů byste měli být schopni toto rozhraní vymyslet.

Úloha nabízí bonusové testy, které zkoušejí další vlastnosti Vaší implementace:

- hodnoty posunutí lze zadávat jako literály s příponami `_day`, `_days`, `_month`, `_months`, `_year` a `_years`. Zápis je zřejmý z druhé poloviny ukázkových testů. Funkcionalitu lze zprovoznit v C++11 přetížením dalších vhodných operátorů. Pokud tento bonus rozhodnete řešit, pak v odevzdávaném zdrojovém souboru zapněte direktivu `#define TEST_LITERALS`, pokud je tato direktiva odpojena, test se přeskakuje a je vždy hodnocen 0 body. Naopak, pokud se tímto bonusem nechcete zabývat, ponechte direktivu beze změn (v komentáři).
- První bonusový test dále zkouší robustnost implementace operátoru pro výstup. Ideální implementace poskytuje správné výsledky pro libovolné nastavení výstupního streamu a zároveň toto nastavení nemění. Ukázka takového chování je na konci rozšířených testů.

- Druhý bonus zkouší rychlost operací +, -, += a rychlost odečítání dvou dat. Naivní řešení s iterací přes všechny dny nemá šanci tímto testem projít, na vstupu jsou zadávaná velká data (roky převyšující 100000).

V úloze uvažujeme standardní (Gregoriánský) kalendář. Data před rokem 1600 považujeme za neplatná, první platné datum je 1.1.1600. Měsíce mají vždy 30 nebo vždy 31 dní, výjimkou je únor, který má 28 dní (nepřestupný rok) nebo 29 dní (přestupný rok). Podle Gregoriánského kalendáře platí:

1. roky nejsou přestupné,
2. s výjimkou let dělitelných 4, které jsou přestupné,
3. s výjimkou let dělitelných 100, které nejsou přestupné,
4. s výjimkou let dělitelných 400, které jsou přestupné,
5. s výjimkou let dělitelných 4000, které nejsou přestupné. (Toto pravidlo ještě oficiálně neplatí, uvažuje se o něm. My jej pro účely této úlohy ale použijeme.)

Tedy roky 1601, 1602, 1603, ... ,2001, 2002, 2003, 2005, ... nejsou přestupné (pravidlo 1), roky 1604, 1608, ..., 2004, 2008, ..., 2096, 2104, ... jsou přestupné (pravidlo 2), roky 1700, 1800, 1900, 2100, 2200, 2300, ... nejsou přestupné (pravidlo 3), roky 1600, 2000, 2400, ..., 3600, 4400, ... jsou přestupné (pravidlo 4) a roky 4000, 8000, ... nejsou přestupné (pravidlo 5).

Za platná považujte data podle Gregoriánského kalendáře počínaje 1.1.1600. Pokud má vzniknout neplatné datum (vytvoření neplatné instance `CDate` nebo v průběhu výpočtů), musí být vyhlášená výjimka `InvalidDateException`. Deklarace této výjimky je zahrnuta v testovacím prostředí, pro její vyvolání stačí příkaz:

```
throw InvalidDateException();
```

Všimněte si, že výjimka je vyhozena přímo hodnotou (není použito `new InvalidDateException();`). Tomu odpovídá i ovladač výjimky, který očekává přímo výjimku a ne ukazatel.

Nápověda

- Testovací prostředí kontroluje hodnoty ve Vašich objektech tím, že si je zašle do výstupního proudu a kontroluje jejich textovou podobu. Bez správně fungujícího operátoru pro výstup bude Vaše řešení hodnoceno velmi nízko (téměř 0 bodů).
- Operátor pro výstup implementujte správně -- neposílejte data na `cout`, posílejte je do předaného výstupního proudu. Za výstupem data do proudu nepřidávejte odřádkování ani jiné bílé znaky.
- Pokud Vám program nejde zkompilovat, ujistěte se, že máte správně přetížené operátory. Zejména si zkontrolujte kvalifikátory `const`.
- Za zamyšlení stojí operátor +=, který musí správně reagovat na vznik neplatného data i v průběhu úprav uvedených na pravé straně. Před návrhem řešení se podívejte na ukázky práce, toto požadované chování má na návrh řešení velký vliv.
- Při implementaci můžete použít `std::vector`, `std::list` a `std::string`. Zbývající kontejnery z STL ale nejsou dostupné.
- Pro vyřešení této úlohy není úplně vhodné používat funkce `mktime` a `localtime`: jejich rozsah obecně nemusí být dostatečný (např. pro data před 1.1.1970, toto je typicky problém pro 32 bitové systémy), jsou potenciálně nebezpečné při neopatrném použití (vliv časových zón) a mohou počítat přestupné roky odlišně od zadání v této úloze (např. mohou rok 4000 považovat za přestupný).
- Při testování bonusového testu - přetížených suffixových operátorů - se nerozlišují verze suffixů v jednotném a množném čísle. Tedy zápisy `1_day` a `1_days` jsou oba považované za správné a ekvivalentní, stejně tak jsou za správné a ekvivalentní považované zápisy např. `12_months` a `12_month`.