Úkolem je realizovat třídy, které implementují správu skladu v supermarketu.

Předpokládáme sklad zboží v supermarketu. V tomto skladu se nachází značné množství zboží. Zboží je jednoznačně identifikováno svým jménem (řetězec). Můžete počítat s tím, že většina jmen výrobků má rozumnou délku (např. do 30 znaků). Dále si pro zboží pamatujeme dobu trvanlivosti a počet naskladněných kusů.

Ve skladu (třídě `CSupermarket`) chceme provádět následující operace, každá operace je implementovaná v metodě:

implicitní konstruktor
: inicializuje prázdnou instanci skladu,

Store (name, expireDate, cnt)
: metoda naskladní zboží. Zboží je identifikované svým jménem (`name`), dobou trvanlivosti (`expireDate`) a počtem kusů (`cnt`). Je možné, že později naskladněné zboží může mít kratší trvanlivost než zboží naskladněné dříve (a to i pro zboží téhož jména).

Sell (list)
: metoda realizuje nákup. Parametrem je nákupní seznam typu `list<pair<string,int>>`, který obsahuje názvy zboží a požadovaný počet. Metoda projde sklad, zjistí dostupnost zboží, upraví počty zboží ve skladu a upraví nákupní seznam. Metodu komplikuje fakt, že zákazníci nepíší krasopisně a software zajišťující OCR nákupních seznamů některé názvy přečte s chybou. Implementace s tím musí počítat, proto bude při hledání zboží používat následující postup:
  - primárně hledá skladě zboží přesně stejného jména (rozlišujeme i malá/velká písmena),
  - pokud neexistuje zboží přesně stejného jména, hledá se zboží, kde se název liší v jednom znaku - překlepu (stále rozlišujeme malá/velká písmena). Pokud se podaří najít právě jedno takové zboží, bude vybráno,
  - pokud se nepodaří nalézt (tedy neexistuje žádné zboží lišící se právě v jednom znaku, nebo existuje více různých zboží lišících se v jednom znaku, nebude se vydávat žádné zboží.

  Pokud zboží nalezne, bude metoda Sell vydávat vždy zboží od nejstaršího (nejdříve se zbavuje zboží s nejmenší dobou trvanlivosti). Pokud je na skladě dostatečné množství zboží, metoda jej vydá a odstraní položku z nákupního seznamu. Pokud na skladě není dostatečné množství zboží, metoda vydá dostupné množství a o toto množství sníží počet v nákupním seznamu.

  Při nakupování postupuje metoda "transakčně": nejprve pro každé zboží na seznamu rozhodne, zda zboží existuje (zda jej nalezne ve skladu, případně jednoznačně nalezne zboží lišící se jen v jednom znaku názvu) a teprve následně začne zboží ze skladu vydávat a aktualizovat nákupní seznam. Toto chování je demonstrované v konci ukázkového běhu, kdy při nákupu koka-koly (seznam `l10`) název `Cake` není jednoznačný, pro následný prodej (po vyprodání `cake`) již ale jednoznačný je. Na ukázce se seznamem `l14` je pak vidět, že nákupní seznam je při výdeji zboží upravován v pořadí od první k poslední položce.

Expired (date)
: metoda zjistí seznam zboží, kterému skončí doba trvanlivosti před zadaným datem. Výsledkem je seznam se jménem zboží a s počtem kusů, kde trvanlivost končí před zadaným datem. Zboží ve výsledném seznamu je seřazené podle počtu kusů sestupně.

Odevzdávejte zdrojový kód s implementací třídy `CSupermarket` a `CDate`. Za základ

implementace použijte přiložený soubor s deklarací metod a se sadou základních testů. Pro implementaci se může hodit doplnit i další pomocné třídy.

V povinném testu je ve skladu rel. malé množství zboží. Povinnými testy by měla projít i lineární implementace. Další testy pracují s mnohem větším objemem dat, kde lineární implementace nestačí. Takové řešení bude velmi citelně penalizované v nepovinném testu. Pro zrychlení využijte asociativní kontejnery z STL.

Asociativní kontejnery lze snadno využít při vyhledávání na přesnou shodu. Nepovinný test pracuje většinou se správnými názvy zboží, tedy řešení rozumně kombinující asociativní a lineární vyhledávání tímto testem projde. Bonusový test pracuje s velkým objemem dat a s velkým množstvím nepřesných jmen zboží. Pro jeho zvládnutí je potřeba použít asociativní kontejnery kreativním způsobem.

Metody `Store` a `Sell` jsou volané často a měly by být efektivní. Metoda `Expires` je volaná mnohem méně často, její efektivita není tak kritická. **POZOR:** pokud je metoda hrubě neefektivní, dokáže způsobit překročení časového limitu. Vracený seznam může být dlouhý, kvadratický algoritmus je pro takovou délku příliš pomalý.

Při implementaci můžete/musíte využít kolekce z STL. Není ale rozumné na všechny vnitřní struktury používat kolekci `vector`. Pokud chcete využívat C++11 kontejnery `unordered_set / unordered_map`, pak hashovací funktor neodvozujte jako specializaci `std::hash`. Hashovací funkci/funktor deklarujte explicitně při vytváření instance `unordered_set / unordered_map`. (Specializace `std::hash` předpokládá opětovné otevření jmenného prostoru `std`. To se těžko realizuje, pokud jste uzavřeni do jiného jmenného prostoru. Návody dostupné na internetu (stack overflow, cpp reference) implicitně předpokládají, že jmenné prostory nepoužíváte, na nich doporučované řešení nejsou ideálně kompatibilní.)