Hledání v sekvencích

Termín odevzdání:	**03.05.2020 23:59:59**
Pozdní odevzdání s penalizací:	**30.06.2020 23:59:59** (Penále za pozdní odevzdání: 100.0000 %)
Hodnocení:	**7.1500**
Max. hodnocení:	**7.1500** (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 vytvořit šablonu třídy `CSearch`. Tato šablona bude sloužit pro vyhledávání výskytu hledané sekvence dat (hledaných sekvencí může být i více než jedna) uvnitř zadané prohledávané sekvence dat. Úkolem je najít všechny výskyty hledaných sekvencí dat v prohledávané sekvenci a vrátit, které hledané sekvence byly nalezené. Šablona bude použitelná např. pro řešení následujících úloh:

- hledáme zadaná klíčová slova (jedno nebo více klíčových slov) v zadaném textu. Tedy hledáme sekvence znaků uvnitř jiné sekvence znaků. Prvkem sekvence je zde znak (`char`), sekvencí je řetězec (`string`),
- hledáme posloupnost čísel (jednu nebo více posloupností) v dlouhém seznamu čísel (prohledávaný seznam čísel). Prvkem sekvence je zde celé číslo (`int`), sekvencí je libovolný sekvenční kontejner STL, tedy například `vector<int>`, `deque<int>`, `forward_list<int>`, `list<int>`,
- hledáme posloupnost (posloupnosti slov) slov v seznamu slov. Prvkem sekvence je zde slovo (`string`), sekvencí je například `vector<string>` nebo `list<string>`.

Aby byla třída `CSearch` co nejuniverzálnější, bude to šablona, která bude parametrizovaná dvěma generickými parametry:

- Prvním parametrem bude datový typ sekvence `_Type`. Může jím být např. zmíněný řetězec `string`, ale stejně dobře i jiný kontejner z STL, konkrétně specializovaný `vector` nebo `list`. Tedy půjde např. vyhledávat v řetězcích, posloupnosti čísel (`vector<int>` nebo `list<int>`), posloupnosti řetězců (`vector<string>` nebo `list<string>`), ... Pozor, protože prohledávanou sekvencí může být i `list`, nesmí se rozhraní `CSearch` spoléhat na náhodný přístup k prvkům sekvence (má k dispozici jen obousměrný iterátor).

  Generickým parametrem `_Type` je jednoznačně určen i typ prvku sekvence. Pro `string` je to datový typ prvku `char`, pro `vector` a `list` pak libovolný primitivní nebo složený datový typ. Obecně o prvku sekvence můžete předpokládat, že má k dispozici kopírující konstruktor, destruktor, operátor přiřazení a operátory pro porovnání na shodu a neshodu (==, !=). Tyto operace jsou buď poskytované kompilátorem (primitivní datové typy), nebo jsou automaticky vygenerované/naprogramované pro složené datové typy. Žádné další rozhraní nemusí být k dispozici. Pozor, obecně není k dispozici ani implicitní konstruktor.

- Druhým nepovinným generickým parametrem šablony je datový typ porovnávače - komparátoru. Jedná se buď o funktor, funkci nebo lambda funkci. Pokud není parametr zadán, předpokládá se standardní komparátor `std::equal_to` pro datový typ prvku sekvence.

Vlastní třída bude mít následující rozhraní:

- Konstruktor s nepovinným parametrem komparátoru. Vytvoří prázdnou instanci třídy vyhledávače.
- Destruktor, pokud bude potřeba
- Kopírující konstruktor a operátor = budou zakázané (viz přiložený základ implementace).
- Metoda `Add (id, needle)`. Metoda přidá další hledanou sekvenci (`needle`) do seznamu vyhledávaných sekvencí. Sekvence je identifikovaná celočíselnou konstantou `id`.
- Metoda `Search(hayHeap)` zkusí v sekvenci `hayHeap` vyhledat dříve zadané hledané sekvence vložené pomocí metody `Add`. Metoda vrátí množinu všech dříve zadaných hledaných sekvencí, které byly v prohledávané sekvenci `hayHeap` nalezené. Vrácená množina bude obsahovat identifikace (`id`) nalezených sekvencí.

Odevzdávejte soubor, který obsahuje implementovanou šablonu třídy `CSearch` a další Vaše podpůrné třídy. 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. 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 jako v ukázce níže.

Při řešení úlohy využijte STL, seznam povolených hlavičkových souborů je obsažen v přiloženém zdrojovém souboru se základem implementace.

Úloha obsahuje povinné a bonusové testy. V bonusovém testu je třída testovaná pro velké množství hledaných sekvencí, které jsou vyhledávané v dlouhém vstupu. Naivní algoritmus bude velmi pomalý. Pro získání bonusu je potřeba implementovat algoritmus rychlejší, například algoritmus Aho-Corasicková.

Základní řešení je po algoritmické stránce velmi přímočaré a krátké. Proto řešení této úlohy nelze použít pro code review. Obtížnost úlohy spočívá ve zvládnutí použité technologie - šablon. Algoritmicky je pak náročnější implementace efektivního algoritmu pro zvládnutí bonusu.

Nápověda:

- Sekvenční kontejnery STL jsou: `vector`, `list`, `deque`, `forward_list`, `array` a `string`. Tyto typy (jejich specializace) mohou být použité jako generický parametr `_Type`.
- Výše uvedené STL kontejnery poskytují užitečné deklarace, které s výhodou využijete. Podívejte se na `_Type::value_type`, `_Type::iterator` a další.
- Budete-li implementovat bonusovou část, je potřeba si předpočítat některá data. Předpočet dat je vhodné provést pouze pokud byl změněn seznam hledaných sekvencí (byla zavolána metoda `Add`), ale ne po každém zavolání metody `Add`. Nabízí se přesunout tento výpočet do metody `Search`. Metoda `Search` je deklarovaná jako `const`, tedy v jejím těle nelze měnit členské proměnné (a tedy ani uložené předpočtené hodnoty). Situaci lze vyřešit, pokud označíte členské proměnné obsahující tato předpočtená data klíčovým slovem `mutable`. Deklarace `mutable` označuje členské proměnné, které lze měnit i v `const` metodách, protože nemění stav objektu (typicky obsahují pouze pomocná data, například jako zde odvozené / předpočtené hodnoty).

Vzorová data: **Download**

☐ **Referenční řešení**

1	**29.04.2020 22:38:12**	**Download**
Stav odevzdání:	Ohodnoceno	
Hodnocení:	7.1500	

- **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: 3.000 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Test nahodnymi daty': Úspěch
    - Dosaženo: 100.00 %, požadováno: 50.00 %
    - Celková doba běhu: 0.121 s (limit: 3.000 s)
    - Úspěch v závazném testu, hodnocení: 100.00 %
  - Test 'Efektivita - rychlost': Program překročil přidělenou maximální dobu běhu
    - Vyčerpání limitu na celý test, program násilně ukončen po: 8.012 s (limit: 8.000 s)
    - Neúspěch v bonusovém testu, hodnocení: Bonus nebude udělen
  - Celkové hodnocení: 100.00 % (= 1.00 * 1.00)
- Celkové procentní hodnocení: 100.00 %
- Celkem bodů: 1.00 * 7.15 = 7.15

SW metriky:

	Celkem	Průměr	Maximum	Jméno funkce
Funkce:	**9**	--	--	--
Řádek kódu:	**109**	**12.11 ± 16.31**	**50**	`main`
Cyklomatická složitost:	**16**	**1.78 ± 1.55**	**6**	`CSearch::Search`