==========
Zadanie słownik, JNP1 C++ 2019/20
==========

Celem tego zadania jest zaimplementowanie wzorca kontenera zachowującego się jak słownik o oczekiwanym czasie wyszukiwania O(1), mającym jednocześnie przewidywalną kolejność iteracji po kluczach, zgodną z kolejnością dodania do słownika. Kontener ten powinien zapewniać możliwie duże gwarancje odporności na wyjątki oraz semantykę kopiowania przy zapisie (ang. copy-on-write).

Kopiowanie przy zapisie to technika optymalizacji szeroko stosowana m.in. w strukturach danych z biblioteki Qt oraz dawniej w implementacjach std::string. Podstawowa jej idea jest taka, że gdy tworzymy kopię obiektu (w C++ za pomocą konstruktora kopiującego lub operator przypisania), to współdzieli ona z obiektem źródłowym wszystkie wewnętrzne zasoby, które mogą w rzeczywistości znajdować się w oddzielnym obiekcie na stercie.
Taki stan rzeczy może trwać do momentu, w którym jedna z kopii musi zostać zmodyfikowana – wtedy modyfikowany obiekt tworzy własną kopię owych zasobów, na których wykonuje modyfikacje.

W ramach tego zadania należy zaimplementować szablon

template <class K, class V, class Hash = std::hash<K>>
class insertion_ordered_map {
};

Szablon ma być sparametryzowany typami K, V i Hash. Można założyć, że typ klucza K ma semantykę wartości, czyli w szczególności dostępne są dla niego bezparametrowy konstruktor domyślny, konstruktor kopiujący, konstruktor przenoszący i operatory przypisania. Od typu K można również wymagać, aby zdefiniowana była funkcja haszująca (parametr Hash) oraz relacja równości (operatory == i !=).
O typie V można jedynie założyć, że ma konstruktor kopiujący.
Parametr Hash powinien być typem obiektu definiującego odpowiedni operator(), a jego domyślną wartością ma być std::hash<K>.

Klasa insertion_ordered_map powinna udostępniać niżej opisane operacje. Przy każdej operacji podana jest jej wymagana złożoność czasowa, gdzie n oznacza liczbę elementów przechowywanych w słowniku. Oczekiwana złożoność czasowa operacji kopiowania przy zapisie powinna wynosić O(n).
Wszystkie operacje muszą zapewniać co najmniej silną odporność na wyjątki, a konstruktor przenoszący i destruktor muszą być no-throw.
Tam, gdzie jest to możliwe i uzasadnione, należy dodać kwalifikator noexcept.

- Konstruktor bezparametrowy tworzący pusty słownik.
  Złożoność czasowa O(1).
  insertion_ordered_map();

- Konstruktor kopiujący. Powinien mieć semantykę copy-on-write, a więc nie kopiuje wewnętrznych struktur danych, jeśli nie jest to potrzebne. Słowniki współdzielą struktury do czasu modyfikacji jednej z nich.
  Złożoność czasowa O(1) lub oczekiwana O(n), jeśli konieczne jest wykonanie

kopii.
insertion_ordered_map(insertion_ordered_map const &other);

- Konstruktor przenoszący.
Złożoność czasowa O(1).
insertion_ordered_map(insertion_ordered_map &&other);

- Operator przypisania. Powinien przyjmować argument przez wartość.
Złożoność czasowa O(1).
insertion_ordered_map &operator=(insertion_ordered_map other);

- Wstawianie do słownika. Jeśli klucz k nie jest przechowywany w słowniku, to wstawia wartość v pod kluczem k i zwraca true. Jeśli klucz k już jest w słowniku, to wartość pod nim przypisana nie zmienia się, ale klucz zostaje przesunięty na koniec porządku iteracji, a metoda zwraca false. Jeśli jest to możliwe, należy unikać kopiowania elementów przechowywanych już w słowniku.
Złożoność czasowa oczekiwana O(1) + ewentualny czas kopiowania słownika.
b**ool insert(K const &k, V const &v);**

- Usuwanie ze słownika. Usuwa wartość znajdującą się pod podanym kluczem k. Jeśli taki klucz nie istnieje, to podnosi wyjątek lookup_error.
Złożoność czasowa oczekiwana O(1) + ewentualny czas kopiowania.
void erase(K const &k);

- Scalanie słowników. Dodaje kopie wszystkich elementów podanego słownika other do bieżącego słownika (this). Wartości pod kluczami już obecnymi w bieżącym słowniku nie są nadpisywane. Klucze ze słownika other pojawiają się w porządku iteracji na końcu, zachowując kolejność względem siebie.
Złożoność czasowa oczekiwana O(n + m), gdzie m to rozmiar słownika other.
void merge(insertion_ordered_map const &other);

- Referencja wartości. Zwraca referencję na wartość przechowywaną w słowniku pod podanym kluczem k. Jeśli taki klucz nie istnieje w słowniku, to podnosi wyjątek lookup_error. Metoda ta powinna być dostępna w wersji z atrybutem const oraz bez niego.
Złożoność czasowa oczekiwana O(1) + ewentualny czas kopiowania.
V &at(K const &k);
V const &at(K const &k) const;

- Operator indeksowania. Zwraca referencję na wartość znajdującą się w słowniku pod podanym kluczem k. Podobnie jak w przypadku kontenerów STL, wywołanie tego operatora z kluczem nieobecnym w słowniku powoduje dodanie pod tym kluczem domyślnej wartości typu V. Oczywiście ten operator ma działać tylko wtedy, jeśli V ma konstruktor bezparametrowy.
Złożoność czasowa oczekiwana O(1).
V &operator[](K const &k);

- Rozmiar słownika. Zwraca liczbę par klucz-wartość w słowniku.
Złożoność czasowa O(1).
size_t size() const;

- Sprawdzenie niepustości słownika. Zwraca true, gdy słownik jest pusty, a false

w przeciwnym przypadku.
Złożoność czasowa O(1).
bool empty() const;

- Czyszczenie słownika. Usuwa wszystkie elementy ze słownika.
Złożoność czasowa O(n).
void clear();

- Sprawdzenie klucza. Zwraca wartość boolowską mówiącą, czy podany klucz k jest w słowniku.
Złożoność czasowa oczekiwana O(1).
bool contains(K const &k) const;

- Klasę iteratora o nazwie iterator oraz metody begin i end, pozwalające przeglądać zbiór kluczy w kolejności ich wstawienia. Iteratory mogą być unieważnione przez dowolną operację modyfikacji zakończoną powodzeniem.
Iterator powinien udostępniać przynajmniej następujące operacje:
- konstruktor bezparametrowy i kopiujący
- operator++ prefiksowy
- operator== i operator!=
- operator* (dereferencji)
Wszystkie operacje w czasie O(1). Przejrzenie całego słownika w czasie O(n).

Iteratory służą jedynie do przeglądania słownika i za ich pomocą nie można go modyfikować, więc zachowują się jak const_iterator z STL.

Klasa lookup_error powinna być zdefiniowana na zewnątrz klasy insertion_ordered_map i powinna dziedziczyć po std::exception.

Klasa insertion_ordered_map powinna być przezroczysta na wyjątki, czyli powinna przepuszczać wszelkie wyjątki zgłaszane przez wywoływane przez nią funkcje i przez operacje na jej składowych, a obserwowalny stan obiektów nie powinien się zmienić. W szczególności operacje modyfikacji zakończone niepowodzeniem nie powinny unieważniać iteratorów.

Rozwiązanie będzie kompilowane za pomocą polecenia

g++ -Wall -Wextra -std=c++17 -O2

i testowane pod kątem prawidłowej obsługi pamięci, np. tak

valgrind --error-exitcode=15 --leak-check=full --show-leak-kinds=all --errors-for-leak-kinds=all --run-cxx-freeres=yes

Rozwiązanie powinno być zawarte w pliku **insertion_ordered_map.h,** który należy umieścić w repozytorium w katalogu

grupaN/zadanie5/ab123456+cd123456

lub

grupaN/zadanie5/ab123456+cd123456+ef123456 ...