|  |  |
| --- | --- |
| Mateusz Woźniak | 13.03.2022 |

**Implementacja słownika w języku c++**

1. **Koncept**

W trakcie pracy wpadłem na cztery możliwe sposoby implementacji słownika:

* Skorzystanie z std::map – odrzuciłem tę opcję, gdyż nie miałbym zbyt wiele do napisania;
* Skorzystanie z std::set i wypełnienia go parami – odrzuciłem, ponieważ od wersji pythona 3.6 słowniki są uporządkowane według kolejności dodania;
* Oparcie słownika o wektor par – zdecydowałem się na tę opcję oraz zaimplementowałem własny pseudo-wektor (ograniczyłem ilość metod i funkcji do tych najbardziej potrzebnych w odniesieniu do słownika), co pozwoliło na napisanie własnych iteratorów;
* Napisanie słownika za pomocą listy wiązanej – niestety późno wpadłem na ten pomysł, a bardzo łatwo porusza się po elementach linked listy.

1. **Klasa MyVector**

Klasa szablonowa MyVector posiada 2 konstruktory oraz destruktor (bezargumentowy oraz taki, który może przechowywać wiele par poprzez skorzystanie z std::initializer\_list). Konstruktor posiadający listę jako argument pozwala na skorzystanie z podobnego zapisu co przy inicjalizacji słownika w pythonie, np. {‘a’: 3, ‘b’: 4} w pythonie odpowiada {std::pair<string,int>(‘a’,3), std::pair<string,int>(‘b’,4)} w mojej implementacji. MyVector posiada 4 metody zarządzające zawartością:

* void push\_back(T elem), która dodaje element na koniec wektora, pamiętając o jego rozszerzeniu w przypadku braku miejsca
* bool empty(), która zwraca true, jeśli wektor jest pusty
* void remove(int position), który usuwa element z podanej pozycji (zrezygnowałem z erase() opartego o iteratory, skoro w pythonie nie da się wprost usunąć kilku par na raz np. przez użycie slice’ów), działa poprzez dodanie nowej tablicy, do której przepisuje w tej samej kolejności elementy przed podaną pozycją, a pozostałe elementy są odpowiednio przesunięte
* void clear(), który zastępuje obecny wektor nowym, pustym

Utworzyłem 4 metody zwracające różne iteratory:

* begin(), ustawiający się na początku wektora
* end(), ustawiający się za ostatnim elementem wektora
* last(), ustawiający się na ostatnim elemencie wektora
* at(int position), ustawiający się na elemencie o indeksie position

Ostatnią ważną częścią klasy MyVector jest zagnieżdżona w niej struktura MyIterator. Przechowuje ona referencję pewien obiekt, a po wywołaniu next()/prev() lub operatorów inkrementacji lub dekrementacji, zmienia swoje położenie na kolejny lub poprzedni obiekt. Klasa posiada również metody has\_next() i has\_prev() zwracające true, gdy istnieje odpowiednio następny lub poprzedni element w wektorze.

1. **Klasa Dictionary**

Podobnie jak MyVector, Dictionary to klasa szablonowa, posiadająca dwa konstruktory – bezargumentowy oraz korzystający z std::initializer\_list. Istnieje możliwość usuwania elementów na podstawie ich klucza (remove\_by\_key(K key)) oraz pozycji (remove\_by\_index()). Metoda dodająca parę klucz-wartość (add(std::pair<K,V> p)) do wektora sprawdza, czy istnieje para o podanym kluczu. Jeśli tak, to nadpisuje wartość, a jeśli nie, to nowa para jest dodana. W dodatku dodałem metody zwracające wektory kluczy i wartości. Operator [] służy jako getter – po podaniu klucza pomiędzy nawiasy, otrzymujemy wartość jemu przypisaną. Większość pozostałych metod wywołuje analogiczne metody z klasy MyVector.