Instrukcja

W czasie pisania programu pamiętaj o:

1. dbaniu o czytelność kodu (odpowiednie formatowanie kodu, nazewnictwo zmiennych adekwatne do ich znaczenia, komentarze),
2. dbaniu o czytelność interfejsu z użytkownikiem (w sposób jawny pytaj użytkownika jakie dane ma podać oraz opisuj wyniki, które zwracasz),
3. przed fragmentem implementującym poszczególne zadania umieść komentarz: `/*Zadanie X */` oraz wypisz na ekranie analogiczny komunikat (X jest numerem zadania): `std::cout << "Zadanie X"<< std::endl;`,
4. każde zadanie umieść w oddzielnej funkcji (w niej dopiero należy odwoływać się do zaimplementowanych funkcji i klas),
5. zaimplementuj menu wyboru zadania, a następnie wykorzystując pętle **do–while** oraz konstrukcję **switch** wykonaj odpowiedni fragment kodu,
6. w zadaniach wymagających udzielenia komentarza bądź odpowiedzi, należy umieścić go w kodzie programu (np. w postaci komentarza albo wydrukować na ekranie),
7. w zadaniach polegających na zaprojektowaniu klasy należy utworzyć jej instancję i wykorzystać zaimplementowaną funkcjonalność.

Wprowadzenie

Kontener std::set

`std::set` jest kontenerem, który implementuje zbiór elementów, tj. żaden z dwóch elementów nie ma takiej samej wartości. `std::set` jest typem szablonowym podobnie jak `std::vector` i jego pierwszym argumentem jest to co chcemy w tym zbiorze przechowywać.

```
std::set<float> liczby;
std::set<std::string> imiona;
std::set<Ocena> oceny;
std::set<Roslina> koszyk;
```

Do zbioru dodawać elementy można na kilka sposobów, np.:

```
float x = 5.0;
liczby.insert(x);
imiona.insert("Wojtek");
oceny.insert(Oceny{"Matematyka", 5.0});
oceny.insert({"Biologia", 5.0});
```

```
        rosliny.emplace(Roslina{TypRosliny::Owoc, "Banan"});
        rosliny.emplace(TypRosliny::Warzywo, "Burak");
```

Zwróć uwagę, że w linii 4 jawnie wywoływany jest konstruktor klasy `Oceny`. W linii 5 konstruktor klasy `Oceny` wywoływany jest natomiast niejawnie. Jest to możliwe, gdyż zbiór `oceny` przechowuje tylko wartości tego typu.

W przypadku przechowywania w zbiorze wartości będących obiektami klas lub struktur, należy zaimplementować operator mniejszości (**operator**<). Jest on wykorzystywany do sprawdzania unikalności poszczególnych elementów. Np.:

```
bool operator<(const Roslina& r1, const Roslina& r2) {
    return r1.nazwa < r2.nazwa;
}
```

Kontener std::map

`std::map` jest implementacją tablicy haszującej, która danemu kluczowi przypisuje jakąś wartość. Przyjmuje dwa argumenty szablonowe z czego pierwszy jest typem klucz, drugi typem wartości przechowywanej.

```
        std::map<std::string, float> cennik;
        std::map<std::string, Koszyk> koszyki_dzieci;
        std::map<int, Oceny> oceny_uczniow;
```

Elementami kontenera `std::map` są obiekty typu `std::pair<Key, Value>`, gdzie klucz i wartość odpowiadają wartościom argumentów szablonowych mapy. `std::pair` jest prostą strukturą z dwoma polami `first` i `second`, przechowujące odpowiednio klucz i wartość.

Wstawianie elementów do mapy wygląda następująco:

```
        std::pair<std::string, float> maslo{"Masło", 3.49};
        cennik.insert(maslo);
        cennik.insert(std::make_pair{"Jogurt", 2.49});
        cennik.insert({"Chleb", 3.0});

5

        koszyki_dzieci.insert({"Marta", Koszyk{}});
```

Algorytmy

Przeglądanie obu kolekcji można realizować na dwa sposoby:

1. za pomocą iteratorów,

```
for (auto it = imiona.begin(); it != imiona.end(), ++it)
    std::cout << *it << std::endl;
```

2. korzystając z pętli for-range.

```
for (auto &ocena : Oceny)
    std::cout << "Przedmiot" << ocena.first << " - ocena: "
            << ocena.second  << std::endl;
```

Większość algorytmów standardowych (http://en.cppreference.com/w/cpp/algorithm) jako pierwsze argumenty przyjmuje iteratory początkowe oraz końcowe zakresu, który chcemy wykorzystać. Kolejnym argumentem jest często jakiś predykat, który w zależności od przeznaczenia algorytmu jest predykatem warunkowym (zwraca wartość prawda-fałsz), albo przekształca element kontenera lub dokonuje innych obliczeń.

Przykładowo algorytm `std::any_of`:

```
template< class InputIt, class UnaryPredicate >
bool any_of( InputIt first, InputIt last, UnaryPredicate p );
```

przyjmuje dwa argumenty będące iteratorami oraz trzeci predykat jednoargumentowy zwracający wartość typu **bool**. Predykatem może być dowolna funkcja, funktor lub wyrażenie lambda spełniające wymagania. Np.:

```
    // sprawdzamy czy istnieje produkt tanszy niz 2 zł.
    std::any_of(cennik.begin(), cennik.end(),
                [](auto &cena) { return cena.second < 2.0; })
```

Zadania

Zadanie 1

Napisz program wczytujący wszystkie słowa z podanego pliku tekstowego do pamięci (np. do `std::vector`). Podczas wczytywania słów usuwaj wszystkie znaki interpunkcyjne oraz zmieniaj wszystkie litery na małe. Wyświetl liczbę wczytanych słów.

Uruchom swój program i wczytaj podany plik tekstowy *macbeth.txt*. Powinieneś wczytać około 18094 słów (ewentualne drobne różnice mogą wynikać ze sposobu tokenizacji tekstu).

Wskazówka 1 Aby usunąć znaki interpunkcyjne z pojedynczego słowa możesz skorzystać z `std::remove_if` oraz `erase`, np.:

```
word.erase(std::remove_if(word.begin(), word.end(), ispunct), word.end())
    ↪ ;
```

Wskazówka 2 Aby zamienić wszystkie znaki w łańcuchu na małe możesz skorzystać z std::transform, np.:

```
std::transform(word.begin(), word.end(), word.begin(), tolower);
```

Zadanie 2

Zmodyfikuj program z poprzedniego zadania w taki sposób by można było wyświetlać liczbę unikalnych słów we wskazanym pliku tekstowym oraz częstość występowania słów wskazanych przez użytkownika.

Uruchom swój program i wczytaj podany plik tekstowy *macbeth.txt*. Powinieneś wczytać unikalnych około 3375 słów. Jak często występuje słowo „Macbeth"? (odp.: ok. 278 razy)

Wskazówka 1 Do zliczania najczęściej występujących słów możesz wykorzystać `std::map`.

Zadanie 3

Zmodyfikuj program z poprzedniego zadania w taki sposób by wyświetlał 20 najczęściej występujących słów w podanym pliku tekstowym (ignorując znaki interpunkcyjne oraz wielkość liter).

Następnie, uruchom swój program i wczytaj plik tekstowy *macbeth.txt*.

Wskazówka 1 Aby wyznaczyć 20 najczęściej występujących słów najprościej skopiować zawartość tablicy haszującej (`std:map`) do listy (`std::vector`), a następnie posortować wynikową listę po drugim elemencie każdej pary klucz-wartość.