POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

PJC Zadania 5

Rozwiązania należy przesłać w postaci odpowiednio podzielonych plików o rozszerzeniach `.hpp` i `.cpp`

Zadanie 1

Stwórz nowy plik nagłówkowy o nazwie `card`, a w nim agregat o tej samej nazwie reprezentujący kartę przypominającą tą z gry Hearthstone, czyli posiadającą koszt (*cost*), nazwę (*name*), opis działania (*description*), siłę ataku (*attack*) i wytrzymałość (*toughness*). Utrzymaj konwencję nazw po angielsku. Następnie napisz funkcję `get_all_cards()`, która zwróci wektor 10 kart o różnych, wybranych przez siebie parametrach. Dodaj funkcję `get_sample_hands()`, która przyjmie przez argument zmienną typu `int` (nazwijmy ją *p*) i zwróci wektor *p* wektorów, gdzie każdy z nich będzie zawierał po 7 kart, czyli reprezentował przykładową rękę. Każda taka ręka powinna składać się z kart z głównego zbioru, czyli z tego zwróconego przez funkcję `get_all_cards()`. Na potrzeby stworzenia ręki możemy założyć, że legalnym jest duplikacja kart (ale ręka nie może zawierać więcej niż 2 kopie takiej samej karty). Postaraj się osiągnąć jakąś losowość zarówno pojedynczej ręki (również będzie to oceniane) jak i całego zbioru przykładowych rąk (czyli również wprowadź jakieś losowanie duplikatów).

Zadanie 2

Stwórz nowy plik nagłówkowy o nazwie `operations` a w nim `enum class` o nazwie `comparing_by` z możliwymi wartościami: `cost`, `name`, `description`, `attack` oraz `toughness`. Następnie napisz funkcję `get_name_comparator()`, która zwróci lambdę porównującą (zgodnie z przedstawionymi na ćwiczeniach założeniami języka) dwa obiekty typu `card` ze względu na ich nazwy (alfabetycznie). Następnie stwórz bliźniaczą funkcję `get_`

`description_comparator()`, która zwróci komparator porównujący obiekty po opisach. Powtórz operację dla kosztu, siły ataku i wytrzymałości karty.

Zadanie 3

Napisz w głównym pliku programu (`main.cpp`) funkcje:

- `sorted()` sortującą przekazany wektor i zwracającą go
- `get_max()` zwracającą największy element
- `get_min()` zwracającą najmniejszy element

Które jako pierwszy argument przyjmą wektor kart, a jako drugi wartość precyzującą kryterium porównywania elementów (typu `comparing_by`). Funkcje te powinny korzystać ze wszystkich przygotowanych w zadaniu 2 elementów.

Przykład:

Następujący kod:

```
for (auto hand : get_sample_hands(5)) {
    for (auto card : sorted(hand, comparing_by::cost)) {
        std::cout << card.cost << ' ' << card.name << '\n';
    }
    std::cout << "========== ### ==========\n";
}
```

Wyświetli 5 razy po 7 linijek (oddzielonych linią widoczną na samym dole) składających się z kosztu i nazwy jakiś kart. Każda z wyświetlanych rąk w powyższym przykładzie ma posortowane karty od najtańszej do najdroższej (koszt).

Przykład 1

Następujący kod:

```
auto most_expensive_card = get_max(
        get_sample_hands(1).front(),
        comparing_by::cost
);

std::cout << most_expensive_card.cost;
```

Wyświetli koszt najdroższej karty wylosowanej w pojedynczej ręce.

Przykład 2