Zadanie 3

Otwarto: poniedziałek, 28 października 2024, 00:00
Wymagane do: niedziela, 17 listopada 2024, 23:59

Turniej rycerski

Wstęp

Rozwiązując to zadanie, studenci powinni poznać:

- rodzaje i dostępność konstruktorów,
- tworzenie operatorów jako metod klasowych i funkcji globalnych,
- operatory porównujące,
- jawne i niejawne konwersje typów,
- listy inicjujące,
- użycie słów kluczowych `const`, `constexpr`, `consteval` i `constinit`,
- użycie słowa kluczowego `inline`.

Polecenie

Zadanie polega na zaimplementowaniu modułu `knights` pozwalającego na rozegranie turnieju rycerskiego o puchar króla lub rękę królewny, odciśniętą rzecz jasna w kamieniu. W ramach tego modułu powinny być udostępnione następujące klasy:

- `Knight` – reprezentująca pojedynczego dzielnego rycerza;
- `Tournament` – reprezentująca turniej rycerski.

Każdy rycerz klasy `Knight` powinien posiadać pewną liczbę sztuk złota na zacny posiłek po trudach walki i zakup pamiątek w wiosce turniejowej. Co jednak ważniejsze, powinien władać bronią i nosić zbroję, których wartość bojowa jest określona pewną ich klasą. Im wyższa klasa broni albo zbroi, tym są one silniejsze. Klasa o zerowej wartości oznacza brak broni albo zbroi. Złoto oraz klasy broni i zbroi są liczbami całkowitymi typu `size_t`.

Klasa `Knight` powinna udostępniać:

- stałą `MAX_GOLD` reprezentującą maksymalną liczbę sztuk złota, które może posiadać rycerz, czyli maksymalną wartość typu `size_t`.
- tworzenie obiektu (rycerza) z trzema parametrami (liczba sztuk złota, klasa broni i klasa zbroi). Nie powinno być możliwości tworzenia obiektu bez podania tych parametrów.
- tworzenie obiektu za pomocą domyślnego kopiowania oraz przenoszenia.
- domyślne operacje przypisania w wersji kopiującej i przenoszącej.
- pozyskanie informacji o złocie trzymanym przez rycerza, o klasie dzierżonej broni i klasie przywdzianej zbroi.
- dorzucenie rycerzowi do sakwy podanej liczby sztuk złota, ale tylko do momentu osiągnięcia dopuszczalnego maksimum.
- zmuszenie rycerza do oddania całego złota, zmiany broni na inną o podanej klasie, oddania całej broni, przebrania się w inną zbroję o podanej klasie oraz oddania zbroi.
- zabranie drugiemu rycerzowi całego jego złota i tylko lepszych (w sensie klasy) składników jego wyposażenia. Tę operację powinien realizować operator `+=`.
- utworzenie nowego rycerza w wyniku operacji „dodawania” (operator `+`) dwóch rycerzy. Nowy rycerz ma mieć tyle złota, ile mają łącznie „dodawani” rycerze, ale maksymalnie `MAX_GOLD` sztuk, oraz lepsze składniki ich wyposażenia.
- porównywanie rycerzy (operatory `<=>` i `==`), które odpowiada na pytanie, kiedy jeden rycerz może pokonać drugiego. Pojedynek (porównanie) wygrywa przede wszystkim ten wojownik, którego broń jest silniejsza od zbroi przeciwnika i którego klasa zbroi jest jednocześnie nie mniejsza niż klasa broni jego przeciwnika. Ponadto, jeśli obaj rycerze mają broń silniejszą od zbroi przeciwnika, to zwycięża ten z lepszą zbroją albo lepszą bronią, gdy zbroje są tej samej klasy. W pozostałych przypadkach będzie remis, co oznacza, że porównywani rycerze są sobie równi.

?

- wypisywanie na standardowe wyjście informacji o rycerzu w postaci `(liczba sztuk złota, klasa broni, klasa zbroi)`.

Należy zapewnić, żeby obiekt klasy `Knight` mógł być używany w wyrażeniach stałych `constexpr`. Ponadto w wyrażeniach stałych powinny być dostępne wszystkie metody, które nie zmieniają stanu obiektu tej klasy. Oprócz tego powinna być dostępna globalna stała `TRAINEE_KNIGHT` inicjowana tylko podczas kompilowania, która reprezentuje praktykanta rycerskiego nieposiadającego żadnego złota oraz mającego broń i zbroję klasy pierwszej.

W drugiej wymaganej w zadaniu klasie `Tournament` powinna być możliwość:

- przechowywania listy pretendentów do tytułu zwycięzcy oraz listy rycerzy wyeliminowanych z turnieju. Kolejność w liście turniejowej określa kolejność, w jakiej rycerze będą stawali do walki. Natomiast rycerze w liście wyeliminowanych są ułożeni w odwrotnej kolejności odpadania z turnieju.
- utworzenia obiektu (turnieju) w oparciu o listę rycerzy przekazaną poprzez parametr. Jeśli podana lista będzie pusta, to jedynym pretendentem w turnieju powinien być praktykant rycerski. Tworzenie turnieju bez podania żadnego parametru nie powinno być możliwe.
- utworzenia obiektu za pomocą kopiowania i przenoszenia oraz realizacji operacji przypisania w wersji kopiującej i przenoszącej.
- dodawania rycerza na koniec listy pretendentów za pomocą operatora `+=`.
- wycofania z turnieju wszystkich zawodników mających wszystkie parametry, czyli złoto, klasę broni i klasę zbroi, identyczne jak podany rycerz, wykorzystując w tym celu operator `-=`.
- rozegrania turnieju. Wynikiem tej operacji powinien być stały iterator do kontenera z pretendentami, wskazujący zwycięzcę lub koniec kontenera (`end()`) w przypadku braku zwycięzcy turnieju. Zasady turnieju są następujące:
    - do walki stają zawsze dwaj pierwsi pretendenci, którzy zdejmowani są z listy turniejowej;
    - walka polega na porównaniu tych dwóch pretendentów;
    - zwycięzca przejmuje złoto przegranego i lepsze składniki jego wyposażenia (broń, zbroję);
    - wygrany wstawiany jest na koniec listy pretendentów;
    - przegrany wstawiany jest do listy wyeliminowanych;
    - w przypadku remisu obaj rycerze zachowują swoje złoto i wyposażenie, ale trafiają do listy wyeliminowanych;
    - turniej kończy się, gdy w liście pretendentów pozostanie jeden (zwycięzca) lub zero (nie ma zwycięzcy) rycerzy;
    - rycerze trafiający do listy wyeliminowanych wstawiani są na jej początek;
    - przy remisie kolejność rycerzy wyeliminowanych pozostaje taka sama, jak była na liście pretendentów.
- uzyskania stałego iteratora oznaczającego brak zwycięzcy turnieju.
- uzyskania informacji o rozmiarze turnieju, czyli łącznej liczbie rycerzy biorących w nim udział (zarówno pretendentów, jak i wyeliminowanych).
- wypisania na standardowym wyjściu aktualnej listy pretendentów oraz rycerzy wyeliminowanych. Wydruk każdego pretendenta powinien być poprzedzony prefiksem `"+ "`, zaś rycerza wyeliminowanego prefiksem `"- "` Koniec wydruku należy oznaczać linią zawierającą znak `'='`.

Na początku każdego rozegrania turnieju lista wyeliminowanych rycerzy powinna być czyszczona. To samo czyszczenie ma się odbywać także w przypadku każdej operacji zmieniającej zawartość listy pretendentów.

Na koniec należy zdefiniować globalną funkcję `max_diff_classes`, pobierającą listę rycerzy i dającą w wyniku parę (`std::pair`) z klasami broni i zbroi, których różnica jest w tej liście największa. Funkcja ta może być wykonywana tylko podczas kompilowania.

Wymagania formalne

Oczekiwane rozwiązanie powinno korzystać z kontenerów i metod udostępnianych przez standardową bibliotekę C++.

Należy ukryć przed światem zewnętrznym wszystkie zmienne globalne i funkcje pomocnicze nienależące do wyspecyfikowanego interfejsu modułu.

Rozwiązanie powinno zawierać plik `knights.h` oraz opcjonalnie plik `knights.cpp`. Pliki te należy umieścić w Moodle. Rozwiązanie będzie kompilowane na maszynie *students* poleceniem

```
g++ -Wall -Wextra -O2 -std=c++20 *.cpp
```

Ocenianie rozwiązania

Ocena automatyczna

Za testy automatyczne zostanie przyznana ocena z przedziału od 0 do 6 punktów. Za błędną nazwę pliku zostanie odjęty 1

?

punkt. Za ostrzeżenia wypisywane przez kompilator zostanie odjęty 1 punkt. Nie ma punktów ułamkowych.

Ocena jakości kodu

Ocena jakości kodu jest z przedziału od 0 do 4 punktów. Nie ma punktów ułamkowych. Odejmujemy punkty za:

- brzydkie formatowanie kodu (niepoprawne wstawianie spacji, wcięć, odstępów, brak komentarzy, magiczne stałe itd.);
- nieprawidłowe, niedostateczne używanie `const`, o ile testy tego nie wykryły;
- zły dobór typu lub kontenera, brak nazw typów, niewiele mówiące nazwy typów;
- rozwlekłą lub nieelegancką strukturę programu, rozpatrywanie zbyt wielu warunków brzegowych, powtarzanie kodu, przechowywanie liczb jako napisów;
- korzystanie z wejścia-wyjścia dostarczanego przez bibliotekę C zamiast ze strumieni;
- nieuzasadnione przekazywanie funkcjom przez wartość dużych obiektów;
- deklarowanie składowych lub metod jako `public`, gdy wystarczyłoby `private` lub `protected`, nadużywanie relacji `friend`;
- niezgodność interfejsu klas z treścią zadania;
- brak użycia operatora `<=>`, definiowanie zbyt wielu operatorów lub zbyt wielu wersji operatorów;
- brak użycia listy inicjującej w konstruktorze;
- wprowadzanie sztucznych ograniczeń na rozmiar danych;
- nieusuwanie lub nieefektywne usuwanie niepotrzebnych już danych;
- brak *header guard*;
- nieukrycie przed światem zewnętrznym wszystkich zmiennych globalnych i funkcji pomocniczych nienależących do wyspecyfikowanego interfejsu modułu;
- użycie `typedef` zamiast `using`;
- inne znalezione i niewymienione w powyższych kryteriach błędy i niezgodności z treścią zadania lub odpowiedziami udzielonymi na forum, a niewykryte przez testy automatyczne;
- niezastosowanie się do uwag udzielonych w poprzednich zadaniach.

Nie wymagamy jeszcze prawidłowego oznaczania metod niezgłaszających wyjątków.

Przykłady użycia

Przykłady użycia modułu `knights` znajdują się w pliku `knights_example.cpp`. Przykłady wydruków wypisywanych przez program `knights_example` znajdują się w pliku `knights_example.out`. Przykłady te są integralną częścią specyfikacji zadania.

Kompilowanie przykładu:

```
g++ -Wall -Wextra -O2 -std=c++20 knights_example.cpp -o knights_example
```

lub

```
g++ -c -Wall -Wextra -O2 -std=c++20 knights_example.cpp
g++ -c -Wall -Wextra -O2 -std=c++20 knights.cpp
g++ knights_example.o knights.o -o knights_example
```

knights_example.cpp	27 października 2024, 11:57
knights_example.out	27 października 2024, 11:57
knights_tests_external.cpp	21 listopada 2024, 19:27
knights_tests.cpp	27 listopada 2024, 16:26