Prosta klasa daty

Kolejne zadanie polega na napisaniu i przetestowaniu klasy `simpleDate` (w sporej części przedstawionej na wykładzie).
Ponieważ temat jest zupełnie nowy i bardzo ważny, zaczniemy tworzenie tej klasy (a raczej i testów, i klasy równocześnie) już na ćwiczeniach.
Jak zwykle będę testować Państwa rozwiązania na własnych, nieco większych testach.

Nie ma żadnych wskazań, dotyczących wewnętrznej struktury klasy, ale nie wydaje mi się, żeby w przypadku tej prostej klasy warto było się porywać na coś innego niż składowe reprezentujące dzień, miesiąc i rok.

Ważne: dla uproszczenia (niekoniecznie, jak się okaże) ograniczymy daty do przedziału od 2020-01-01 do 2023-12-31.

Interfejs klasy – konstruktory

Chciałbym móc tworzyć obiekty typu `simpleDate` w następujący sposób:

```
simpleDate deflt;       // 2020-01-01
simpleDate jan2020(13); // 2020-01-13
simpleDate mar20220(30, 3);  // 2020-03-30
simpleDate dec2021(31, 12, 2021);  // 2021-12-31
```

W przypadku niepoprawnych argumentów (tzn. wartości dnia, miesiąca i roku nie dających daty z interesującego nas przedziału) proszę wygenerować wyjątek `invalid_argument`.

Z tego co widać, wszystkie argumenty w konstruktorze regularnej daty, mają domyślne wartości. Możemy zatem załatwić 4 różne konstruktory, które widać wyżej, jedną funkcją z odpowiednimi parametrami domyślnymi.

`simpleDate` – utworzenie klasy

Zaczniemy standardowo od utworzenia projektu (`zad_2`) i zmiany nazwy pliku głównego (`zad_2.cpp`).

Tworzymy od razu plik nagłówkowy (`simpleDate.h`), w którym będzie definicja klasy i plik źródłowy (`simpleDate.cpp`), w którym umieścimy implementację klasy.

Zaczniemy od zapisania pustej klasy i sprawdzenia, czy wszystko należycie się kompiluje.

Dokładamy teraz konstruktor z parametrami dzień, miesiąc, rok (w tej właśnie kolejności) i z wartościami domyślnymi odpowiednio: 1 1 2020. Intuicyjnie chyba czuć, że dzień, miesiąc i rok powinny mieć swoje miejsce w klasie – dokładamy sekcję `private` z odpowiednimi składowymi.

Teraz można zaimplementować konstruktor (w `cpp`).

I dołożyć metodę `print()` wyświetlającą datę na konsoli.

`simpleDate` – pierwsze testy

Można byłoby chyba pierwsze testy napisać?

Problem w tym, że na razie sprawdzenie, czy wszystko jest ok, wymaga wpatrywania się w konsolę. Utwórzmy dwa obiekty `simpleDate` w funkcji `main`, wyświetlmy ich zawartość i kontynuujmy budowanie klasy.

W szczególności dołóżmy funkcje odczytu wartości dnia, miesiąca i roku:

```
int month() const;
int day() const;
int year() const;
```

Te funkcje są tak proste, że typowo znajdziemy ich implementację bezpośrednio w definicji klasy (w pliku nagłówkowym).

Mając te metody, mogę zautomatyzować testy, no i zmodyfikować `print()` (swoją drogą, czy `print()` jest `const`?).

`simpleDate` – trochę więcej testów

Z metodami dostępu do dnia, miesiąca i roku można pokusić się o automatyzację testów. O ile poprawne daty nie stanowią problemu, to stwierdzenie, że data jest niepoprawna – już tak.

Po pierwsze, konstruktor na niepoprawne argumenty powinien zareagować wyjątkiem (trzeba go przejąć w programie).

Po drugie, sytuacji błędnych jest całkiem sporo:

- wartości zupełnie spoza zakresu (czyli jakie?)
- wartości ogólnie dopuszczalne, ale dające w zestawieniu niepoprawną datę (znowu: jakie?)

Zacznijmy tworzyć te testy (3-4 przypadki), żeby mieć gotową infrastrukturę. Dopisywanie kolejnych testów pójdzie dalej sprawnie.

Proszę zwrócić uwagę na to, gdzie w implementacji pojawia się Wam nieco więcej kodu. W większości przypadków te pierwsze fragmenty kodu bardzo szybko powinny zmienić swoje położenie (refaktoryzacja kodu).

`simpleDate` – generowanie wyjątku

W konstruktorze obiektów klasy simpleDate możemy dostać argumenty, które dają niepoprawną datę lub datę spoza dopuszczalnego zakresu.

Ponieważ mamy w planach pomocniczą funkcję is_valid rozbiegówka konstruktora może wyglądać następująco:

```
simpleDate(int d = 1, int m = 1, int y = 2020)
{
   if (!is_valid(d, m, y))
      throw std::runtime_error("Invalid date.");
   // reszta konstruktora ...
}
```

Klasa wyjątku `runtime_error` jest zdefiniowana w pliku nagłówkowym `stdexcept`.

`simpleDate` – przechwytywanie wyjątku

Wyjątek wygenerowany a nie przechwycony spowoduje zakończenie wykonania programu, co radykalnie zredukuje nam możliwości testowania sytuacji niepoprawnych. Powinniśmy zatem przy testach konstruktora przechwytywać wyjątki.

Można to zrobić w następujący sposób:

```
try
{
    simpleDate invalid(32, 1, 2020);
    std::cerr << "Missing exception for invalid date.\n";
}
catch (std::runtime_error& ex)
{
    // to jest ok - tego sie spodziewamy
    std::cerr << ex.what() << '\n';
}
```

simpleDate – operacje na datach

W tej pierwszej wersji klasy zaimplementujemy tylko dwie funkcje operujące na datach:

```
void next_day(); // przesuń datę o dzień w przyszłość
void prev_day(); // przesuń datę o dzień wstecz
```

Funkcje wyglądają niewinnie i wydaje się, że ich implementacja zajmie chwilę, ale to niestety złudzenia.

Zacznijmy od przemyślenia kwestii testów. Co będzie potrzebne, żeby przetestować te funkcje?

Może przydałaby się możliwość tworzenia kopii istniejącej daty:

```
simpleDate copy(mar2020);
```

oraz podstawiania

```
copy = dec2020;
```

(to oczywiście podpucha – kompilator już to za nas zrobił).

simpleDate – operacje na datach

Przy modyfikacji daty – nawet o jeden dzień – musimy sobie odpowiedzieć na pytanie: co zrobić, jeśli operacja zmiany daty „wyprowadzi” datę poza dopuszczalny zakres?

Mamy dwie możliwości:

1. Wygenerować wyjątek. To jest zgodne z podejściem zastosowanym w konstruktorze, gdzie nie dopuszczamy tworzenia daty spoza dopuszczalnego zakresu.
2. Nie wykonać operacji. Przy operacjach jednodniowych oznacza to pozostanie na ostatnim (`next_day`) lub pierwszym (`prev_day`) dniu dopuszczalnego zakresu.

Jeśli wybierzemy drugie rozwiązanie będzie nieco prościej z testowaniem – zatem przyjmujemy je jako obowiązujące.

simpleDate – podsumowanie

W klasie `simpleDate` mamy do zaimplementowania:

```
simpleDate(int d = 1, int m = 1, int y = 2020);
int day() const;
int month() const;
int year() const;
void print() const;

// modyfikacje daty
void next_day();
void prev_day();

// funkcje pomocnicze
bool is_leap_year(int year) const;
bool is_valid(int day, int month, int year) const;
int days_in_month(int month, int year) const;
```

Dostarczanie rozwiązania

Rozwiązaniem zadania są pliki **`simpleDate.h`**, (definicja klasy) i **`simpleDate.cpp`** (definicje metod klasy).

Proszę załadować w `Moodle` tylko te dwa pliki (oczywiście na początku każdego pliku trzeba umieścić w komentarzu imię, nazwisko i nr albumu autora).

Rozwiązanie należy przekazać do:

23 listopada 2022 23:59