# Zadanie trzecie

Zadanie polega na napisaniu i przetestowaniu klasy `gi_date`, w sporej części zgodnej z opisaną na wykładzie. Tym razem dość solidne testy są dołączone do zadania, ale warto też korzystać z własnych, może mniej masowych, ale łatwiejszych do ogarnięcia.
Trzeba jednak zwracać uwagę na precyzyjne trzymanie się wymagań interfejsu.

**Ważne:** trzymamy się przedziału od `2020-01-01` do `2023-12-31`, ale tym razem jest ściśle określony sposób wewnętrznej reprezentacji daty: jedna liczba całkowita oznaczająca liczbę dni od początku przedziału reprezentowanych dat. Ja 1 stycznia 2020 reprezentowałbym jako 0 (ale nic nie narzucam – to w końcu część prywatna).

# Interfejs klasy (i nie tylko)

```
class gi_date
{
public:
    explicit gi_date(int day = 1, int month = 1, int year = 2020);
    int get_year() const;
    int get_month() const;
    int get_day() const;
    void next_day();
    void prev_day();
    gi_date& operator+=(int days);
    gi_date& operator-=(int days);
    static bool is_leap_year(int year);
    static bool is_valid(int day, int month, int year);
    static int days_in_month(int month, int year);

private:
    int days_since_start;
     // ...
};

bool operator==(const gi_date& left, const gi_date& right);
bool operator!=(const gi_date& left, const gi_date& right);
bool operator<(const gi_date& left, const gi_date& right);
int operator-(const gi_date& left, const gi_date& right);
std::ostream& operator<<(std::ostream& os, const gi_date& right);
```

# Komentarz

W metodach zmieniających datę nie można przekroczyć zakresu; zamiast tego należy ustawić najpóźniejszą lub najwcześniejszą dopuszczalną datę.

Operator – daje różnicę (w dniach) pomiędzy dwoma datami.

Zwracam uwagę na to, że definicje interfejsu klasy `gi_date` oraz funkcji pomocniczych są całkiem sensowne i nie należy ich zmieniać. Należy natomiast uzupełnić definicję klasy o składowe prywatne.

Przy ocenianiu będzie istotna nie tylko liczba błędów, ale także jakość kodu:

- sensowne użycie i definicje metod prywatnych,
- rozsądne podejście do testowania własnego kodu,
- jednolite i konsekwentne stosowanie wcięć.

# Testowanie we własnym zakresie

Doświadczenie z zadania drugiego pokazuje, że traktujecie Państwo prowadzącego jako automat testujący. Po części jest to naturalne, bo nie macie Państwo jeszcze wprawy w pisaniu testów. Ale nie nabierzecie wprawy nie pisząc testów.

Żeby Was do tego zachęcić wprowadzam cennik usług testowania:

(i) wykrycie błędu po raz pierwszy

– 0 punktów,

(ii) wykrycie tego samego błędu po raz drugi i kolejny

– 1 punkt.

# Dostarczanie rozwiązania

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

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:

**30 listopada 2022 23:59**