Lista 6 – klasy i operatory

1. Na na końcu strony https://en.cppreference.com/w/cpp/language/operators można znaleźć gotowy szkielet klasy `Fraction` reprezentującej ułamki.

   a) Zamień w niej typ wartości przechowywanych w liczniku i mianowniku na `int64_t` (lub `long long int`), a następnie dodaj implementację następujących funkcji:

   - `operator +`
   - `operator +=`
   - `operator -`
   - `operator -=`
   - `operator /`
   - `operator /=`
   - funkcja składowa `to_double()` zwracająca numeryczną wartość ułamka jako liczbę typu `double`.

   b) Na podstawie punktu a) napisz i przetestuj funkcje, które wyznaczą kilka kolejnych wartości następujących sum jako dokładne ułamki oraz jako ich wartości numeryczne:

   i. $\sum_{i=1}^{N} \frac{1}{i(i+1)}, N=1,\ldots,100.$ (granica: 1)

   ii. $\sum_{i=1}^{N} \frac{(-1)^{N+1}}{i}, N=1,\ldots,20.$ (granica: ln 2)

   iii. $\sum_{i=1}^{N} \frac{1}{2^i i}, N=1,\ldots,15.$ (granica: ln 2)

2. Ułamki łańcuchowe opisane są m.in., tu: https://pl.wikipedia.org/wiki/U%C5%82amek_%C5%82a%C5%84cuchowy i tu: https://en.wikipedia.org/wiki/Continued_fraction Niech `F(n, x)` oznacza ułamek łańcuchowy, który powstaje po obcięciu do pierwszych `n` wyrazów pełnego ułamka łańcuchowego dla liczby x (zakładamy że ułamek łańcuchowy dla `x` jest nieskończony). Wyznacz `F(n,x)` dla `n = 1,…,10` jako ułamek i jako liczbę typu `double`, dla:

   a) $x=\frac{\sqrt{5}+1}{2}$ (złoty podział, ułamek łańcuchowy: [1;1,1,1,1,1,….])

   b) $x=e$ (podstawa logarytmów naturalnych, ułamek łańcuchowy: [2;1,2,1,1,4,1,1,6,1,1,8,1,1,10,…])

Uwagi.
Początek rozwiązania zad 1.b.ii może wyglądać następująco:

```
i = 1, 1/1    1
i = 2, 1/2    0.5
i = 3, 5/6    0.833333
i = 4, 7/12   0.583333
i = 5, 47/60  0.783333
```

Dla zdania 2b):

```
2/1           2
3/1           3
8/3           2.6666666666666665
11/4          2.75
19/7          2.7142857142857144
```

Pytania kontrolne

1. Dlaczego złożone operatory przypisania ( `+=, -=, <<=`, etc) zaleca się definiować w klasie, a dwuargumentowe operatory arytmetyczne – poza klasą?
2. Czy funkcja `Fraction::gcd` z cppreference korzysta z wartości obiektu, na którym działa? Jeśli nie, to czy można jej nadać atrybut `static`, czyli `static int64_t` **`gcd`**`(int64_t a, int64_t b)`, bez konieczności modyfikowania reszty programu?

3. Co modyfikator `static` oznacza w przypadku funkcji składowych klas?
4. W implementacji cppreference wskaż przynajmniej 2 miejsca, w których implementacja pewnej funkcji korzysta z implementacji innej funkcji.
5. `operator*` zaimplementowany został (w cppreference) niepokojąco prosto: `return lhs *= rhs;` Jaką właściwość przeciążonego operatora `*=` tu zastosowano? (Wskazówka: to jest łatwe pytanie).
6. Mniej więcej w połowie zalinkowanej strony znajduje się rozdział *Increment and decrement*, a w nim przykłady przeciążania operatorów `++` i `--`. W jaki sposób rozróżnia się wersję przedrostkową (`++x`) od przyrostkowej (`x++`). Która wersja inkrementacji dowolnego obiektu `x` wydaje się w ogólnym przypadku prostsza i bardziej efektywna: `++x` czy `x++`?
7. Nieco wcześniej w dokumentacji cppreference znajduje się rozdział *Function call operator*. Rzuć okiem przynajmniej na pierwszy przykład. Załóżmy, że w klasie `X` zdefiniowano funkcję składową o sygnaturze `double operator()(double x, double y) const;` i że masz obiekt `x` klasy `X`. Jak tę funkcję można najłatwiej wywołać na obiekcie `x`? Czy widzisz jakieś zastosowania?
8. Co oznacza modyfikator `const` użyty po nawiasach zawierających listę parametrów funkcji składowej w klasie, np w: `double operator()(double x, double y) const;`?
9. Co oznacza słowo kluczowe `this` użyte w omawianym tu kodzie z serwisu cppreference?
10. W klasie `Fraction` nie zdefiniowano konstruktora kopiującego ani operatora przypisania. Sprawdź lub uwierz na słowo, że mimo tego możesz ich używać. Jak działają domyślne wersje tych funkcji, wygenerowane przez kompilator?