Zadanie 1 (na 3.0). Napisz klasę ułamka zwykłego, zawierającą dwa pola prywatne (licznik i mianownik), oraz trzy konstruktory (bez-, jedno- i dwuargumentowy), zgodnie z laboratorium 01. Dodaj do niego operatory dodawania, odejmowania, mnożenia, dzielenia oraz postinkrementacji, preinkrementacji, postdekrementacji i predekrementacji (`a++, ++a, a--, --a`). Po każdej operacji tego wymagającej uprość ułamek. Dodaj unarny operator - przestawiający znak ułamka na przeciwny. Dodaj operator ! odwracający ułamek (licznik staje się mianownikiem, a mianownik licznikiem).
Napisz odpowiedni operator umożliwiający ładne wypisanie ułamka na ekran. Zaprzyjaźnij go z klasą ułamka zwykłego.

```
int main(int, char**)
    {
    Fraction a=Fraction(1, 2)+Fraction(1, 4);
    Fraction b=a-Fraction(1, 8);
    Fraction c=b*Fraction(1, 2);
    Fraction d=c/Fraction(10, 2);

    std::cout << a << "\n" << b << "\n" << c << "\n" << d << "\n";
    std::cout << -d << "\n";
    std::cout << !d << "\n";
    std::cout << !-d << "\n";
    std::cout << !-!++---!-!-Fraction(1, 10) << "\n";

    Fraction e(3, 4);
    std::cout << "wartosc poczatkowa: " << e << "\n";
    std::cout << "preinkrementacja: " << ++e << "\n";
    std::cout << "po operacji: " << e << "\n";
    std::cout << "predekrementacja: " << --e << "\n";
    std::cout << "po operacji: " << e << "\n";
    std::cout << "postinkrementacja: " << e++ << "\n";
    std::cout << "po operacji: " << e << "\n";
    std::cout << "postdekrementacja: " << e-- << "\n";
    std::cout << "po operacji: " << e << "\n";

    // Zadziała?
    //std::cout << a+2 << "\n";

    return 0;
    }
```

Zadanie 2 (na 4.0). Zmodyfikuj klasę stosu z laboratorium 3. tak, aby przechowywała ułamki zwykłe zamiast wektorów. Dodaj operator [] zwracający dowolny element ze stosu oraz operator bool (operator rzutowania na typ bool) zwracający prawdę jeśli na stosie jest przynajmniej jeden element.
Napisz dwie funkcje przyjmujące referencję na stos:

1. ściągająca dwa elementy z wierzchu stosu i wrzucająca na wierzch ich sumę,
2. ściągająca jeden element z wierzchu stosu i wrzucająca na wierzch jego dwukrotność.

Zaprezentuj działanie:

```
int main(int, char**)
    {
    Stack s;

    s.push(Fraction(2));
    s.push(Fraction(3));
    s.push(Fraction(4));

    // Co zostanie wypisane?
    //Fraction &a=s[0];
    //Fraction &b=a;
    //b=Fraction(5);
    //std::cout << a << b << s[0];

    std::cout << "Na stosie mamy: \n";
    for(int i=0; i<s.getSize(); ++i)
                std::cout << " " << i << ". " << s[i] << "\n";

    stackAdd(s);
    stackMul(s);
    stackAdd(s);

    std::cout << "Wynik: " << s.pop() << "\n";

    return 0;
    }
```

Zadanie 3 (na 5.0). Do klasy ułamka zwykłego dodaj operatory porównania. Stwórz stos 20 ułamków zwykłych. Stwórz mapę (std::map) ułamka zwykłego na liczbę całkowitą. Wpisz do mapy liczbę wystąpień każdego ułamka w stosie:

```
#include <cstdlib>
#include <iostream>
#include <map>

[...]

int main(int, char**)
        {
        Stack s;

        for(int i=0; i<20; ++i)
                s.push(Fraction((rand()%7)-3));

        std::map<Fraction, int> histogram;

        while(s)
                ++histogram[s.pop()];

        for([...])
                std::cout << [...] << ": " << [...] << "\n";

        return 0;
        }
```

Utwórz wektor (std::vector) ułamków zwykłych. Posortuj jego zawartość przy pomocy std::sort rosnąco po wartościach bezwzględnych ułamków. Konieczne może okazać się zrobienie tzw. funktora (obiektu klasy zawierającej przeciążony operator()). Wypisz posortowany wektor.

```
#include <cstdlib>
#include <iostream>
#include <vector>

[...]

int main(int, char**)
        {
        std::vector<Fraction> array;

        for(int i=0; i<20; ++i)
                array.push_back(Fraction((rand()%7)-3));

        [...] // sortowanie

        for([...])
                std::cout << [...] << "\n";

        return 0;
        }
```