**­**

**Лабораторна робота №1**

З дисципліни «ООП»

«Розробка простого класу»

Виконав: Землянський Едуард

Група: КВ-22

**Постановка задачі**

Розробити клас згідно до варіанту. Не використовувати default-реалізацію

конструкторів та операторів (реалізувати самостійно). Обробка помилкових

ситуацій не є обов’язковою в рамках даної роботи. Пропонується самостійно

розробити інтерфейс класу, зокрема, виходячи з власних міркувань зручності

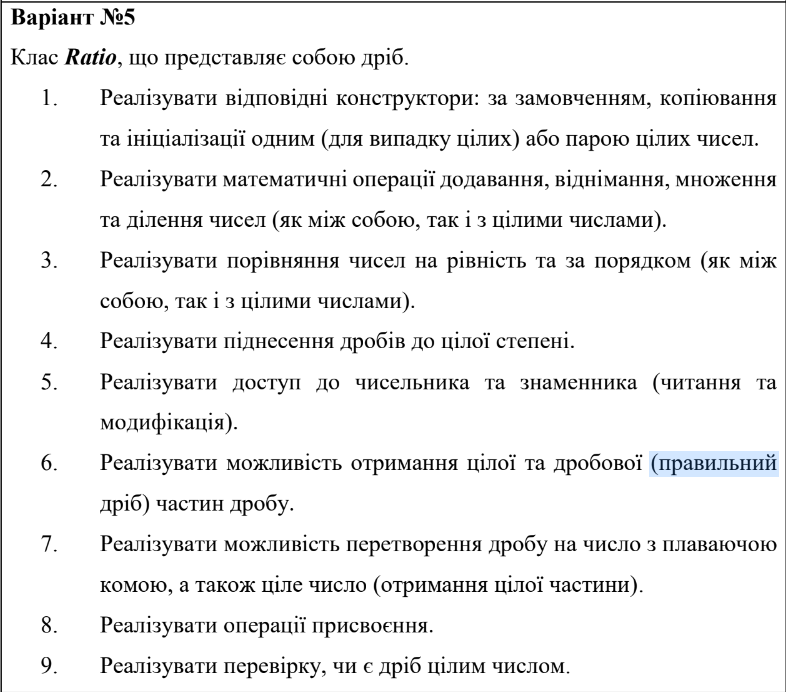
його подальшого використання (бажано зробити використання класу по

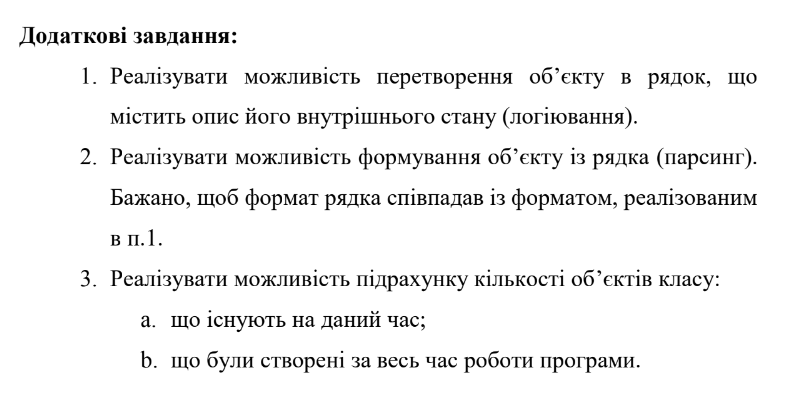
можливості інтуїтивно зрозумілим).

**Варіант завдання**

№5

**Вимоги до програми**

****

****

**Текст програми**

#include <iostream>

#include <iomanip>

#include <string>

#include <cstring>

#include <string>

#include <cmath>

using namespace std;

// Variant 5 Zemlyanskyi Eduard KV-22

int InstanceCounter = 0;

int AllTimeInstanceCounter = 0;

bool showLogs = false;

class Ratio{

    public:

        int numerator;

        int denominator;

        Ratio();

        Ratio(int numerator, int denominator = 1);

        void SetRatio(int numerator, int denominator);

        void GetRatio();

        float GetRatioAsFloat();

        void CopyRatio(const Ratio & refRatio);

        void Simplify();

        void Compare(const Ratio & refRatio);

        void CompareToInt(int ref\_numerator);

        void operator=(const Ratio & refRatio);

        void operator=(int integer);

        void CheckIfInt();

        int GetIntegerPart();

        Ratio GetRatioPart();

        Ratio Power(int integer);

        Ratio operator+(const Ratio & refRatio);

        Ratio operator+(int integer);

        Ratio operator-(const Ratio & refRatio);

        Ratio operator-(int integer);

        Ratio operator\*(const Ratio & refRatio);

        Ratio operator\*(int integer);

        Ratio operator/(const Ratio & refRatio);

        Ratio operator/(int integer);

        ~Ratio();

};

Ratio::Ratio(){

    if (showLogs){

        cout << "Constructor without parameters called for " << this << endl;

    }

    InstanceCounter++;

    AllTimeInstanceCounter++;

}

Ratio::Ratio(int numerator, int default\_denominator) : denominator(default\_denominator){

    if (showLogs){

        cout << "Constructor with parameters called for " << this << endl;

    }

    SetRatio(numerator, denominator);

    InstanceCounter++;

    AllTimeInstanceCounter++;

}

void Ratio::SetRatio(int passed\_numerator, int passed\_denominator){

    if (showLogs){

        cout << "Setter called for " << this << endl;

    }

    numerator = passed\_numerator;

    denominator = passed\_denominator;

}

void Ratio::GetRatio(){

    if (showLogs){

        cout << "Getter called for " << this << endl;

    }

    cout << "Ratio: " << numerator << "/" << denominator << endl;

}

float Ratio::GetRatioAsFloat(){

    if (showLogs){

        cout << "Float getter called for " << this << endl;

    }

    // cout << "Float: " << float(numerator)/float(denominator) << endl;

    return float(numerator)/float(denominator);

}

void Ratio::CopyRatio(const Ratio & refRatio){

    if (showLogs){

        cout << "Copying called for " << this << endl;

    }

    numerator = refRatio.numerator;

    denominator = refRatio.denominator;

}

void Ratio::Simplify(){

    if (showLogs){

        cout << "Simplifying called for " << this << endl;

    }

    if(numerator % denominator == 0){

        numerator /= denominator;

        denominator = 1;

    }

    else if (denominator % numerator == 0){

        denominator /= numerator;

        numerator = 1;

    }

    else {

        while(numerator % 2 == 0 && denominator % 2 == 0){

            numerator /= 2;

            denominator /= 2;

        }

    }

}

int Ratio::GetIntegerPart(){

    if (showLogs){

        cout << "Getting integer part called for " << this << endl;

    }

    int integer;

    if(numerator > denominator){

        // cout << "Integer part: " << integer << " Ratio part: " << numerator - integer \* denominator << "/" << denominator << endl;

        return int(numerator/denominator);

    }

    else if (numerator < denominator){

        cout << "No integer part can be taken." << endl;

    }

    return 0;

}

Ratio Ratio::GetRatioPart(){

    if (showLogs){

        cout << "Getting ratio part called for " << this << endl;

    }

    int new\_numerator, integer;

    if(numerator > denominator){

        int integer = GetIntegerPart();

        new\_numerator = numerator - integer \* denominator;

    }

    else if (numerator < denominator){

        cout << "No integer part can be taken." << endl;

    }

    Ratio newRatio(new\_numerator, denominator);

    return newRatio;

}

void Ratio::Compare(const Ratio & refRatio){

    if (showLogs){

        cout << "Comparison called for " << this << endl;

    }

    int new\_numerator1, new\_numerator2;

    if (denominator == refRatio.denominator){

        if (numerator > refRatio.numerator){

            cout << numerator << "/" << denominator << " > " << refRatio.numerator << "/" << refRatio.denominator << endl;

        }

        else{

            cout << numerator << "/" << denominator << " < " << refRatio.numerator << "/" << refRatio.denominator << endl;

        }

    }

    else{

        new\_numerator1 = numerator \* refRatio.denominator;

        new\_numerator2 = refRatio.numerator \* denominator;

        if (new\_numerator1 > new\_numerator2){

            cout << numerator << "/" << denominator << " > " << refRatio.numerator << "/" << refRatio.denominator << endl;

        }

        else{

            cout << numerator << "/" << denominator << " < " << refRatio.numerator << "/" << refRatio.denominator << endl;

        }

    }

}

void Ratio::CompareToInt(int integer){

    if (numerator > integer \* denominator){

        cout << numerator << "/" << denominator << " > " << integer << endl;

    } else{

            cout << numerator << "/" << denominator << " < " << integer << endl;

        }

    if (showLogs){

        cout << "Comparison called for " << this << endl;

    }

}

void Ratio::operator=(const Ratio & refRatio){

    CopyRatio(refRatio);

}

void Ratio::operator=(int integer){

    CopyRatio(Ratio(integer, 1));

}

void Ratio::CheckIfInt(){

    cout << "Ratio " << numerator << "/" << denominator;

    Simplify();

    if (denominator == 1){

        cout << " is an integer: " << numerator << endl;

    } else {

        cout << " is not an integer." << endl;

    }

}

Ratio Ratio::operator+(const Ratio & refRatio){

    int new\_numerator, new\_denominator;

    if (denominator == refRatio.denominator){

        new\_numerator = numerator + refRatio.numerator;

        new\_denominator = denominator;

    }

    else{

        new\_numerator = numerator \* refRatio.denominator + refRatio.numerator \* denominator;

        new\_denominator = denominator \* refRatio.denominator;

    }

    Ratio newRatio = Ratio(new\_numerator, new\_denominator);

    newRatio.Simplify();

    if (showLogs){

        cout << "Addition called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio Ratio::operator+(int integer){

    Ratio newRatio = Ratio(numerator + integer \* denominator, denominator);

    newRatio.Simplify();

    if (showLogs){

        cout << "Addition called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio Ratio::operator-(const Ratio & refRatio){

    int new\_numerator, new\_denominator;

    if (denominator == refRatio.denominator){

        new\_numerator = numerator - refRatio.numerator;

        new\_denominator = denominator;

    }

    else{

        new\_numerator = numerator \* refRatio.denominator - refRatio.numerator \* denominator;

        new\_denominator = denominator \* refRatio.denominator;

    }

    Ratio newRatio = Ratio(new\_numerator, new\_denominator);

    newRatio.Simplify();

    if (showLogs){

        cout << "Substraction called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio Ratio::operator-(int integer){

    Ratio newRatio = Ratio(numerator - integer \* denominator, denominator);

    newRatio.Simplify();

    if (showLogs){

        cout << "Substraction called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio Ratio::operator\*(const Ratio & refRatio){

    int new\_numerator, new\_denominator;

    new\_numerator = numerator \* refRatio.numerator;

    new\_denominator = denominator \* refRatio.denominator;

    Ratio newRatio = Ratio(new\_numerator, new\_denominator);

    newRatio.Simplify();

    if (showLogs){

        cout << "Multiplication called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio Ratio::operator\*(int integer){

    Ratio newRatio = Ratio(numerator \* integer, denominator);

    newRatio.Simplify();

    if (showLogs){

        cout << "Multiplication called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio Ratio::operator/(const Ratio & refRatio){

    Ratio newRatio = this->operator\*(Ratio(refRatio.denominator, refRatio.numerator));

    if (showLogs){

        cout << "Dividing called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio Ratio::operator/(int integer){

    Ratio newRatio = this->operator\*(Ratio(1, integer));

    if (showLogs){

        cout << "Dividing called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio Ratio::Power(int integer){

    Ratio newRatio = Ratio(pow(numerator, integer), pow(denominator, integer));

    newRatio.Simplify();

    if (showLogs){

        cout << "Power elevation called for " << &newRatio << endl;

    }

    return newRatio;

}

Ratio::~Ratio(){

    if (showLogs){

        cout << "Destructor called for " << this << endl;

    }

    InstanceCounter--;

}

void AskLogs(){

    string Answer;

    cout << "Show logs ? (y/n): ";

    cin >> Answer;

    if ((Answer == "y" || Answer == "yes" || Answer != "0") && Answer != "no" && Answer != "n" && Answer != "nope"){

        showLogs = true;

    }

}

Ratio ParseRatio(string RefString){

    int numerator = stoi(RefString);

    int denominator = stoi(RefString.substr(RefString.find('/') + 1));

    return Ratio(numerator, denominator);

}

int main(){

    AskLogs();

    // test here

    return 0;

}

**Тестування програми**

1. Реалізувати відповідні конструктори: за замовченням, копіювання та ініціалізації одним (для випадку цілих) або парою цілих чисел.



1. Реалізувати математичні операції додавання, віднімання, множення та ділення чисел (як між собою, так і з цілими числами).

Ratio ratio1 = Ratio(1, 2), ratio2 = Ratio(1, 2), ratio3;

    ratio3 = ratio1 + ratio2;

    ratio1.GetRatio();

    ratio2.GetRatio();

    ratio3.GetRatio();

    cout << "---" << endl;

    ratio1 = Ratio(6, 3); ratio2 = Ratio(1, 3);

    ratio3 = ratio1 - ratio2;

    ratio1.GetRatio();

    ratio2.GetRatio();

    ratio3.GetRatio();

    cout << "---" << endl;

    ratio1 = Ratio(4, 2); ratio2 = Ratio(1, 3);

    ratio3 = ratio1 \* ratio2;

    ratio1.GetRatio();

    ratio2.GetRatio();

    ratio3.GetRatio();

    cout << "---" << endl;

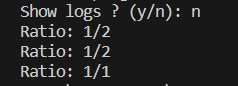
    ratio1 = Ratio(1, 2); ratio2 = Ratio(1, 3);

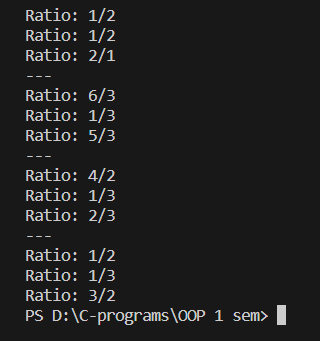
    ratio3 = ratio1 / ratio2;

    ratio1.GetRatio();

    ratio2.GetRatio();

    ratio3.GetRatio();





З цілим числом:

Ratio ratio1 = Ratio(1, 2), ratio2 = Ratio(3), ratio3;

    ratio3 = ratio1 + ratio2;

    ratio3.GetRatio();

    ratio3 = ratio1 - ratio2;

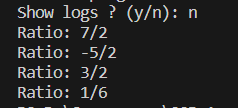
    ratio3.GetRatio();

    ratio3 = ratio1 \* ratio2;

    ratio3.GetRatio();

    ratio3 = ratio1 / ratio2;

    ratio3.GetRatio();

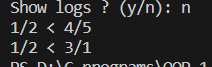


1. Реалізувати порівняння чисел на рівність та за порядком (як між собою, так і з цілими числами).

Ratio ratio1 = Ratio(1, 2), ratio2 = Ratio(3), ratio3;

    ratio1.Compare(Ratio(4, 5));

    ratio1.Compare(ratio2);

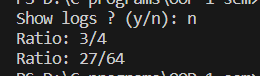


1. Реалізувати піднесення дробів до цілої степені.

Ratio ratio1 = Ratio(1, 2), ratio2 = Ratio(3, 4), ratio3;

    ratio2.GetRatio();

    ratio2.Power(3).GetRatio();



1. Реалізувати доступ до чисельника та знаменника (читання та модифікація).

Ratio ratio1 = Ratio(1, 2), ratio2 = Ratio(3, 4), ratio3;

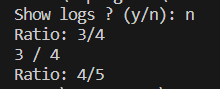
    ratio2.GetRatio();

    cout << ratio2.numerator << " / " << ratio2.denominator << endl;

    ratio2.numerator = 4;

    ratio2.denominator = 5;

    ratio2.GetRatio();

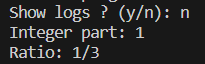


1. Реалізувати можливість отримання цілої та дробової (правильний дріб) частин дробу.

Ratio ratio1 = Ratio(1, 2), ratio2 = Ratio(4, 3), ratio3;

    cout << "Integer part: " << ratio2.GetIntegerPart() << endl;

    ratio2.GetRatioPart().GetRatio();



1. Реалізувати можливість перетворення дробу на число з плаваючою комою, а також ціле число (отримання цілої частини).

Ratio ratio1 = Ratio(1, 2), ratio2 = Ratio(4, 3), ratio3;

    cout << ratio2.GetRatioAsFloat() << endl;



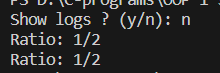
1. Реалізувати операції присвоєння.

Ratio ratio1 = Ratio(1, 2), ratio2 = Ratio(4, 3), ratio3;

    ratio3 = ratio1;

    ratio1.GetRatio();

    ratio3.GetRatio();



1. Реалізувати перевірку, чи є дріб цілим числом.

Ratio ratio1 = Ratio(12, 2), ratio2 = Ratio(4, 3), ratio3 = Ratio(3);

    ratio1.CheckIfInt();

    ratio2.CheckIfInt();

    ratio3.CheckIfInt();

