#include "Rational.h"

//конструктор вызывается при создании объекта

// деструктор - при выхождении переменной из области видимости

int main() {

setlocale(0, "");

/\*

//Rational r;

//Rational r1[5]; // пятикратное создание объекта, используется конструктор без параметров

Rational r(1, 3);

Rational r2;

// r.print();

r2 = r.add(r);

// r2.print();

r2 = r + r2;

//std::cout << r2;

\*/

std::vector<Rational> arr;

std::ifstream fin("text.txt");

Rational ra;

/\*while (!fin.eof())

{

fin >> ra;

arr.push\_back(ra);

}\*/

std::istream\_iterator<Rational> it\_in(fin), it\_cur; //потоковый итератор

std::copy(it\_in, it\_cur, std::back\_inserter(arr));

/\*

for (size\_t i = 0; i < arr.size(); i++)

{

std::cout << arr[i];

}\*/

std::copy(arr.begin(), arr.end(), std::ostream\_iterator<Rational>(std::cout, "\n"));

return 0;

}

#pragma once

#include <iostream>

#include <fstream>

#include <vector>

#include <iterator>

class Rational

{

public:

Rational(size\_t numer\_= 0, size\_t denom\_ = 1); //констреуктор с параметрами, которые заданы значениями по умолчанию

// будет работать и в качестве конструктора без параметров

Rational(const Rational&);

~Rational();

const Rational operator+(Rational r);

size\_t getNumer() const { return numer; }

void setNumer(size\_t n) { numer = n; }

size\_t getDenom() const { return denom; }

void setDenom(size\_t n) { denom = n; }

void print();

const Rational add(const Rational&) const;

friend std::istream& operator>>(std::istream&, Rational& r);

private:

size\_t numer, denom;

void reduce();

};

size\_t GCD(size\_t a, size\_t b);

std::ostream& operator<<(std::ostream&, const Rational& r);

#include "Rational.h"

Rational::Rational(size\_t numer\_, size\_t denom\_)

{

numer = numer\_;

denom = denom\_;

std::cout << "Constructor with parametrs\n";

}

Rational::Rational(const Rational& r) //конструктор копирования

{

numer = r.numer;

denom = r.denom;

std::cout << "Copy constructor\n";

}

const Rational Rational::add(const Rational &r) const

{

Rational sum;

sum.numer = numer \* r.denom + r.numer \* denom;

sum.denom = denom \* r.denom;

sum.reduce();

return sum;

}

const Rational Rational::operator+(Rational r)

{

Rational sum;

sum.numer = numer \* r.denom + r.numer \* denom;

sum.denom = denom \* r.denom;

sum.reduce();

return sum;

}

Rational::~Rational()

{

std::cout << "Destructor\n";

}

/\*

void Rational::print()

{

std::cout << numer << "/" << denom << "\n";

}

\*/

void Rational::reduce()

{

size\_t gcd = GCD(numer, denom);

if (gcd != 0) {

numer /= gcd;

denom /= gcd;

}

}

size\_t GCD(size\_t a, size\_t b)

{

if (a == 0 || b == 0)

{

return 0;

}

while (a != b)

{

if (a > b)

{

a -= b;

}

else

{

b -= a;

}

}

return a;

}

std::ostream& operator<<(std::ostream& out, const Rational& r)

{

out << r.getNumer() << "/" << r.getDenom();

return out;

}

std::istream& operator>>(std::istream& in, Rational& r)

{

in >> r.numer >> r.denom;

return in;

}

#include "pch.h"

#include "..\Rational.h"

#include "..\Rational.cpp"

TEST(TDefaultConstructor, TestConstr1) {

Rational r;

EXPECT\_EQ(r.getNumer(), 0);

EXPECT\_EQ(r.getDenom(), 1);

}

TEST(TConstructor, TestConstr2) {

Rational r(1,7);

EXPECT\_EQ(r.getNumer(), 1);

EXPECT\_EQ(r.getDenom(), 7);

}

TEST(TestSum, TestReduce) {

Rational r(1, 2), r1(1, 2);

Rational sum;

sum = r1.add(r);

EXPECT\_EQ(sum.getNumer(), 1);

EXPECT\_EQ(sum.getDenom(), 1);

}

TEST(TestSum, TestZero)

{

Rational r(0, 2), r1(0, 2);

Rational sum;

sum = r1.add(r);

EXPECT\_EQ(sum.getNumer(), 0);

EXPECT\_EQ(sum.getDenom(), 4);

}

// Train.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include "TrainClass.h"

int main()

{

TRAIN main;

TRAIN train1("Minsk", 1, false);

TRAIN train2("Brest", 2, true);

TRAIN train3("Mogilev", 3, true);

TRAIN train4(train1);

TRAIN train5("Polotsk", 4, false);

std::vector<TRAIN> trains = { train1, train2, train3, train4, train5 };

std::string destination;

std::cout << "Enter the destination ";

std::cin >> destination;

main.printVectorOfTrainsToDestination(trains, destination);

main.printVectorOfExpress(trains, destination);

return 0;

}

// Запуск программы: CTRL+F5 или меню "Отладка" > "Запуск без отладки"

// Отладка программы: F5 или меню "Отладка" > "Запустить отладку"

// Советы по началу работы

// 1. В окне обозревателя решений можно добавлять файлы и управлять ими.

// 2. В окне Team Explorer можно подключиться к системе управления версиями.

// 3. В окне "Выходные данные" можно просматривать выходные данные сборки и другие сообщения.

// 4. В окне "Список ошибок" можно просматривать ошибки.

// 5. Последовательно выберите пункты меню "Проект" > "Добавить новый элемент", чтобы создать файлы кода, или "Проект" > "Добавить существующий элемент", чтобы добавить в проект существующие файлы кода.

// 6. Чтобы снова открыть этот проект позже, выберите пункты меню "Файл" > "Открыть" > "Проект" и выберите SLN-файл.

#pragma once

#include <iostream>

#include <string>

#include <vector>

#include <sstream>

class TRAIN {

private:

std::string destination;

int numberOfTrain;

bool isExpress;

std::vector<TRAIN> CreateVectorOfTrainsToDestination(std::vector<TRAIN>, std::string);

std::vector<TRAIN> CreateVectorOfExpress(std::vector<TRAIN>, std::string);

public:

TRAIN();

TRAIN(std::string, int, bool \_isExpress = false); //constructor of initialization

TRAIN(const TRAIN&); //constructor of copy

std::string trainToString() const;

//void TrainsHeadingToTheDestination(std::vector<TRAIN>, std::string);

void printTrains(std::vector<TRAIN>);

void printVectorOfTrainsToDestination(std::vector<TRAIN>, std::string);

void printVectorOfExpress(std::vector<TRAIN>, std::string);

std::string getDestination() { return destination; }

void setDestination(std::string newDestination) { destination = newDestination; }

int getNumerOfTrain() { return numberOfTrain; }

void setDestination(int newNumerOfTrain) { numberOfTrain = newNumerOfTrain; }

bool getIsExpress() { return isExpress; }

void setIsExpress() { isExpress = !isExpress; }

};

#include "TrainClass.h"

TRAIN::TRAIN() {

destination = "";

numberOfTrain = -1;

isExpress = false;

}

TRAIN::TRAIN(std::string \_destination, int \_numberOfTrain, bool \_isExpress)

{

destination = \_destination;

numberOfTrain = \_numberOfTrain;

isExpress = \_isExpress;

//std::cout << "Constructor of initialization\n";

}

TRAIN::TRAIN(const TRAIN& oldTrain)

{

destination = oldTrain.destination;

numberOfTrain = oldTrain.numberOfTrain;

isExpress = oldTrain.isExpress;

//std::cout << "Copy constructor\n";

}

/\*

void TRAIN::TrainsHeadingToTheDestination(std::vector<TRAIN> trains, std::string \_destination)

{

for (size\_t i = 0; i < trains.size(); i++)

{

if (trains[i].destination.compare(\_destination)) {

std::cout << "\n";

std::cout << trains[i].trainToString();

std::cout << "\n";

}

}

}\*/

std::vector<TRAIN> TRAIN::CreateVectorOfTrainsToDestination(std::vector<TRAIN> trains, std::string \_destination)

{

std::vector<TRAIN> result;

for (size\_t i = 0; i < trains.size(); i++)

{

if (!trains[i].destination.compare(\_destination)) {

result.push\_back(trains[i]);

}

}

return result;

}

std::vector<TRAIN> TRAIN::CreateVectorOfExpress(std::vector<TRAIN> trains, std::string \_destanation)

{

std::vector<TRAIN> result;

std::vector<TRAIN> trainsSoughtForDestination = CreateVectorOfTrainsToDestination(trains, \_destanation);

for (size\_t i = 0; i < trainsSoughtForDestination.size(); i++)

{

if (trainsSoughtForDestination[i].isExpress) {

result.push\_back(trainsSoughtForDestination[i]);

}

}

return result;

}

std::string TRAIN::trainToString() const {

std::string number;

std::ostringstream oss;

oss << numberOfTrain;

number = oss.str();

std::string \_isExpress = isExpress ? "yes" : "no";

/\*

std::string result = "\n\tThe destination: " + destination + "\n\tThe number of train: " + number +

"\n\tIs the train express: " + \_isExpress + "\n";

\*/

std::string result = "\n\tThe train number " + number + "\n\tis running to " + destination +

"\n\tIs the train express: " + \_isExpress + "\n";

return result;

}

void TRAIN::printTrains(std::vector<TRAIN> trains)

{

for (size\_t i = 0; i < trains.size(); i++)

{

std::cout << "\n";

std::cout << trains[i].trainToString();

std::cout << "\n";

}

}

void TRAIN::printVectorOfTrainsToDestination(std::vector<TRAIN> trains, std::string \_destination)

{

std::vector<TRAIN> result = CreateVectorOfTrainsToDestination(trains, \_destination);

if (result.size() != 0) {

std::vector<TRAIN> result = CreateVectorOfTrainsToDestination(trains, \_destination);

std::cout << "\nTo " << \_destination << " go the next trains:\n";

printTrains(result);

}

else {

std::cout << "\nTo " << \_destination << " no trains running\n";

}

}

void TRAIN::printVectorOfExpress(std::vector<TRAIN> trains, std::string \_destination)

{

std::vector<TRAIN> result = CreateVectorOfExpress(trains, \_destination);

if (result.size() != 0) {

std::cout << "\nTo " << \_destination << " go the next expresses:\n";

printTrains(result);

}

else {

std::cout << "\nTo " << \_destination << " no expresses running\n";

}

}