ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ЛАБОРАТОРНОЙ РАБОТЕ №1

По дисциплине «Языки программирования»

ВАРИАНТ 30(29)

Выполнил: ст. гр. ТКИ-142

Костин Роман Сергеевич

Проверил: к.т.н., доц. Васильева М. А.

(Проверил: к.т.н, доц. Балакина Е. П.)

Москва 2024

Оглавление

[1. 1 Формулировка задания 3](#_Toc168131603)

[1. 2 UML диаграмма 4](#_Toc168131604)

[1. 3 Код программы на языке C++ 5](#_Toc168131605)

[1. 4 Результаты выполнения программы 11](#_Toc168131606)

[1. 5 Тесты классов 12](#_Toc168131607)

[1. 6 Отметка о выполнении задания в веб-хостинге системы контроля версий 14](#_Toc168131608)

* 1. Формулировка задания

Разработать библиотеку классов.

Класс **Длина дуги** содержит две точки, заданные географическими координатами(широта от −90° до +90°, долгота от −180° до +180°)которые, в свою очередь, являются типами **Угловая** **мера.** Предусмотреть создание объекта, принимающего как тип географическая координата, так и вещественное число). Реализовать метод расчета длины. Предусмотреть вывод информации (сериализация в строку ToString) об объекте в стандартный поток вывода и метод чтения из стандартного потока ввода.

Класс **Угловая** **мера** (Angle) хранит информацию о градусах (degrees), минутах (minutes) и секундах (seconds). Градусы и минуты представлены целым числом, а секунды – десятичной дробью. Предусмотреть конструкторы, принимающие величину угловой меры в секундах; величину угловой меры в виде тройки значений градусы, минуты, секунды. При невозможности создать объект – минуты и секунды должны быть представлены неотрицательными числами не более 60, выбросить исключение. Переопределить операторы ввода и вывода (сдвига вправо и влево, соответственно), операторы равенства и неравенства.

Для расчета длины между двумя географическими объектами можно использовать формулу <https://en.wikipedia.org/wiki/Great-circle_distance>.

* 1. UML диаграмма

UML диаграмма для классов Angle и Arc\_length представлена ниже (Рисунок 1).

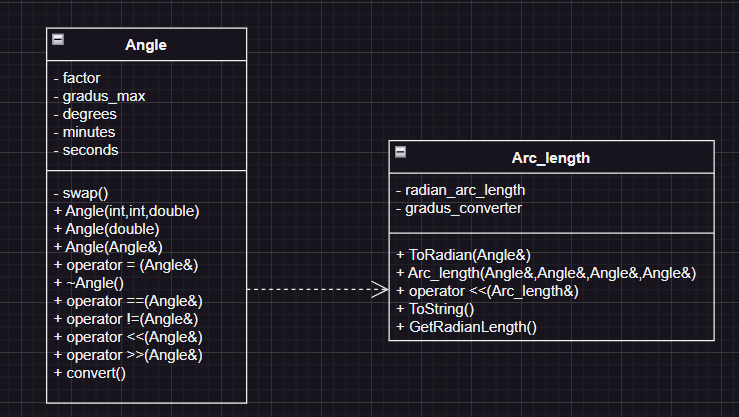


Рисунок 1 - UML диаграмма для классов Угловая мера и Длина дуги

* 1. Код программы на языке C++

main.cpp:

#include <iostream>

#include "..\solver\Angle.h"

#include "..\solver\Arc\_length.h"

int main(void)

{

Angle:: Angle a(10,0,0);

Angle::Angle b(a);

Angle::Angle c(20,0,0);

Angle::Angle d(20,0,0);

Arc::Arc\_length arclength(a,b,c,d);

std::cin >> a;

std::cout << a << std::endl << d<<std::endl << arclength;

return 0;

}

Angle.h:

#pragma once

#include <limits>

#include <iostream>

namespace Angle

{

class Angle

{

private:

/\*

\*@brief значение для перевода секунд и минут в градусы

\*/

static const int factor = 60;

/\*

\*@brief максимальное значение градуса

\*/

static const int gradus\_max = 360;

/\*

\*@brief значение в градусах

\*/

int degrees{ 0 };

/\*

\*@brief значение в минутах

\*/

int minutes{ 0 };

/\*

\*@brief значение в секундах

\*/

double seconds{ 0 };

/\*

\*@brief функция swap для класса Angle

\*/

void swap(Angle& angle);

public:

/\*

\*@brief конструктор для случая, когда введены все значения об угле

\*/

Angle(int degr, int min, double sec);

/\*

\*@brief конструктор для случая, когда введено только значение в секундах, которые переводятся в минуты, а затем в градусы

\*/

Angle(double sec);

/\*

\*@brief конструктор копирования

\*/

Angle(Angle& A) = default;

/\*

\*@brief оператор "=" для класса Angle

\*/

Angle& operator = (const Angle& angle) = default;

/\*

\*@brief деструктор класса Angle

\*/

~Angle() = default;

/\*

\*@brief оператор "==" для класса Angle

\*/

bool operator == (const Angle& angle) const;

/\*

\*@brief оператор "!=" для класса Angle

\*/

bool operator != (const Angle& angle) const;

/\*

\*@brief оператор "<<" для класса Angle

\*/

friend std::ostream& operator<<(std::ostream& output,const Angle& angle);

/\*

\*@brief оператор ">>" для класса Angle

\*/

friend std::istream& operator>>(std::istream& input, Angle& angle);

/\*

\*@brief функция для перевода градусной меры в вещественное значение, необходимое для подсчётов синусов, косинусов и т.д.

\*/

double convert() const;

};

}

Angle.cpp:

#include <iostream>

#include "Angle.h"

#include "cmath"

Angle::Angle::Angle(int degr, int min, double sec)

{

if (min<0 || sec < 0 || min > factor || sec > factor||std::abs(degr)>gradus\_max)

{

throw std::invalid\_argument("wrong angle!");

}

this->degrees = degr;

this->minutes = min;

this->seconds = sec;

}

Angle::Angle::Angle(double sec)

{

int negative = 1;

if (sec < 0)

{

sec \*= -1;

negative = -1;

}

int temp\_min = sec / factor;

this->degrees = (negative\*(temp\_min/factor)%gradus\_max);

this->minutes = temp\_min%factor;

this->seconds = fmod(sec,factor);

}

bool Angle::Angle::operator==(const Angle& angle) const

{

return(this->degrees == angle.degrees && this->minutes == angle.minutes && std::abs(this->seconds - angle.seconds) <= std::numeric\_limits<double>::epsilon());

}

bool Angle::Angle::operator!=(const Angle& angle) const

{

return !(\*this == angle);

}

double Angle::Angle::convert() const

{

double temp = (double)this->degrees + (double)this->minutes / factor + this->seconds / pow(factor,2);

return temp;

}

std::ostream& Angle::operator<<(std::ostream& output, const Angle& angle)

{

output << angle.degrees << " degrees " << angle.minutes << " minutes " << angle.seconds << " seconds ";

return output;

}

std::istream& Angle::operator>>(std::istream& input, Angle& angle)

{

int degrees = 0,

minutes = 0;

double seconds = 0;

input >> degrees >> minutes >> seconds;

Angle temp(degrees,minutes,seconds);

temp.swap(angle);

return input;

}

void Angle::Angle::swap(Angle& angle)

{

std::swap(this->degrees,angle.degrees);

std::swap(this->minutes,angle.minutes);

std::swap(this->seconds,angle.seconds);

}

Arc\_length.h:

#pragma once

#include "..\solver\Angle.h"

#include <numbers>

namespace Arc

{

class Arc\_length

{

private:

/\*

\*@brief длина дуги между двумя точками на сфере в радианах

\*/

double radian\_arc\_length;

/\*

\*@brief константа для перевода в радианы

\*/

static constexpr double gradus\_converter = (std::numbers::pi\_v<double>)/180 ;

/\*

\*@brief метод класса длина дуги, который возвращает значение в радианах

\*/

double ToRadian(const Angle::Angle& degree) const;

public:

/\*

\*@brief конструктор класса длина дуги

\*@param first\_point\_longitude - долгота первой точки

\*@param first\_point\_latitude - широта первой точки

\*@param second\_point\_longitude - долгота второй точки

\*@param second\_point\_latitude - широта второй точки

\*/

Arc\_length(const Angle::Angle& first\_point\_longitude,const Angle::Angle& first\_point\_latitude,const Angle::Angle& second\_point\_longitude,const Angle::Angle& second\_point\_latitude);

/\*

\*@brief оператор "<<" для класса длина дуги

\*/

friend std::ostream& operator<<(std::ostream& output, const Arc\_length &arc);

/\*

\*@brief ToString для класса длина дуги

\*/

std::string ToString() const;

/\*

\*@brief геттер для arc\_length

\*@return значение в радианах

\*/

double GetRadianLength() const;

};

}

Arc\_length.cpp:

#include "Arc\_length.h"

#include "cmath"

#include <sstream>

double Arc::Arc\_length::ToRadian(const Angle::Angle& degree) const

{

return degree.convert()\*gradus\_converter;

}

Arc::Arc\_length::Arc\_length(const Angle::Angle& first\_point\_longitude, const Angle::Angle& first\_point\_latitude, const Angle::Angle& second\_point\_longitude, const Angle::Angle& second\_point\_latitude)

{

this->radian\_arc\_length = acos(sin(ToRadian(first\_point\_latitude))\*sin(ToRadian(second\_point\_latitude))+cos(ToRadian(first\_point\_latitude))\*cos(ToRadian(second\_point\_latitude))\*(std::abs(ToRadian(first\_point\_longitude) -ToRadian(second\_point\_longitude))));

}

std::string Arc::Arc\_length::ToString() const

{

std::stringstream buffer{};

buffer << this->radian\_arc\_length;

return buffer.str();

}

double Arc::Arc\_length::GetRadianLength() const

{

double temp = this->radian\_arc\_length;

return temp;

}

std::ostream& Arc::operator<<(std::ostream& output, const Arc\_length& arc)

{

return output << arc.ToString();

}

* 1. Результаты выполнения программы

Результаты выполнения программы представлены ниже (Рисунок 2,Рисунок 3).

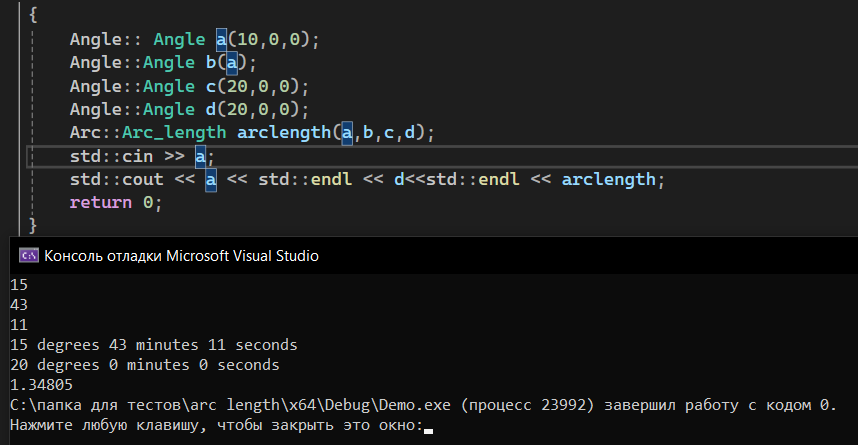


Рисунок 2 – Результат выполнения программы при корректном вводе

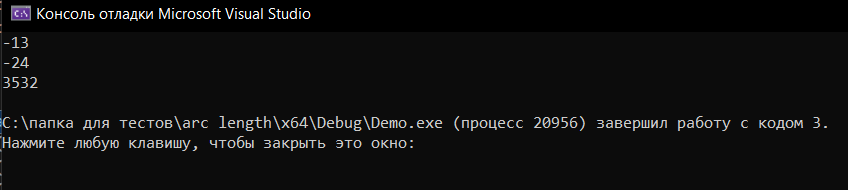


Рисунок 3 – Результат выполнения программы при некорректном вводе

* 1. Тесты классов

UnitTest1.cpp:

#include "pch.h"

#include "CppUnitTest.h"

#include "cmath"

#include "..\solver\Angle.h"

#include "..\solver\Arc\_length.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace AngleTest

{

TEST\_CLASS(AngleTest)

{

public:

TEST\_METHOD(operator\_equal\_equal)

{

int deg = 1, min = 1;

double sec = 1.1;

Angle::Angle a(deg, min, sec);

Angle::Angle b(deg,min,sec);

Assert::IsTrue(a==b);

}

TEST\_METHOD(operator\_not\_equal)

{

Angle::Angle a(1,1,1);

Angle::Angle b(2,2,2);

Assert::IsTrue(a!=b);

}

TEST\_METHOD(seconds\_to\_degrees)

{

Angle::Angle a(-3601);

Angle::Angle b(-1, 0, 1);

Assert::IsTrue(a==b);

}

TEST\_METHOD(seconds\_equal\_360\_degrees)

{

Angle::Angle a(12960000); //3600\*3600

Angle::Angle b(0);

Assert::IsTrue(a == b);

}

TEST\_METHOD(seconds\_value\_is\_more\_than\_360\_degrees)

{

Angle::Angle a(12960001); //3600\*3600+1

Angle::Angle b(1);

Assert::IsTrue(a == b);

}

};

TEST\_CLASS(Arc\_length\_test)

{

TEST\_METHOD(Arc\_lenght\_equal\_to\_double)

{

Angle::Angle flo(0);

Angle::Angle fla(0);

Angle::Angle slo(0);

Angle::Angle sla(0);

Arc::Arc\_length actual(flo, fla, slo, sla);

double radian = (std::numbers::pi\_v<double>) / 180;

double expected = acos(sin(0 \* radian) \* sin(0 \* radian) + cos(0 \* radian) \* cos(0 \* radian) \* (std::abs(0 \* radian - 0 \* radian)));

Assert::AreEqual(actual.GetRadianLength(), expected);

}

};

}

Результаты выполнения тестов классов представлены ниже (Рисунок 4).

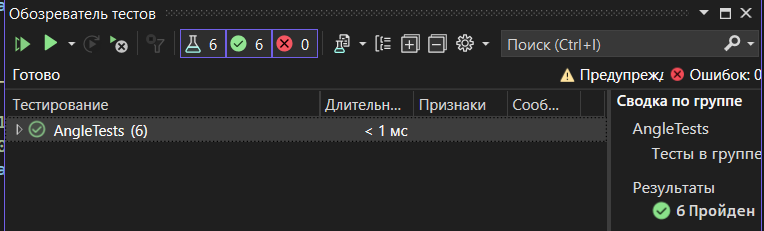


Рисунок 4 – Результат выполнения тестов

* 1. Отметка о выполнении задания в веб-хостинге системы контроля версий

В веб-хостинге системы контроля версий GitHub были поставлены approve-ы. Они представлен ниже (Рисунок 5, Рисунок 6, Рисунок 7).

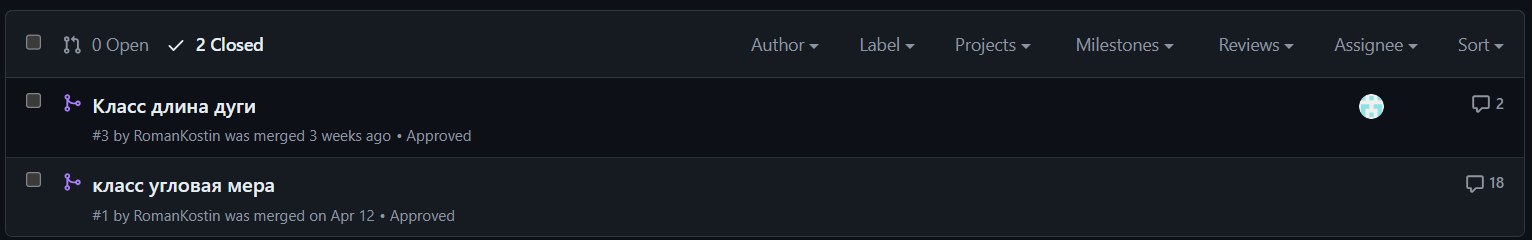


Рисунок 5 – Approve на классы Длина дуги и Угловая мера

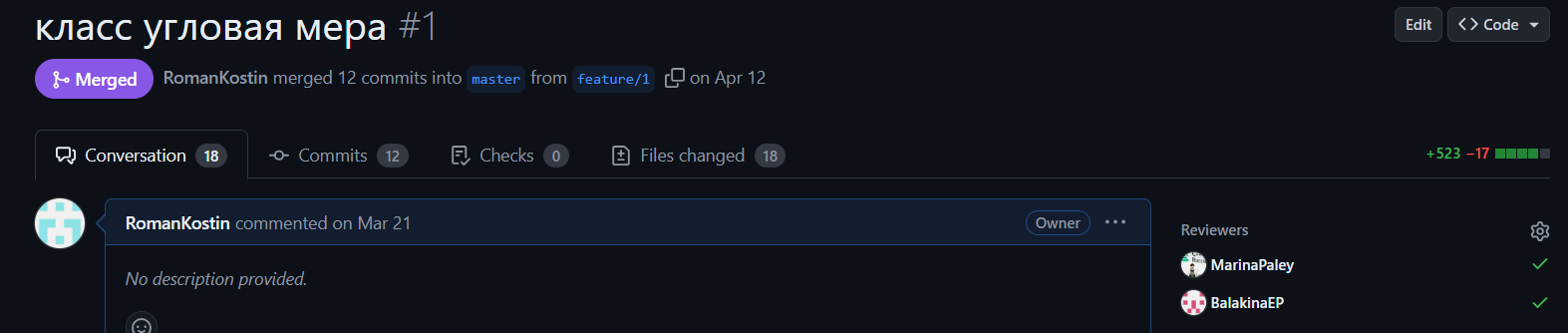


Рисунок 6 – Approve на класс Угловая мера

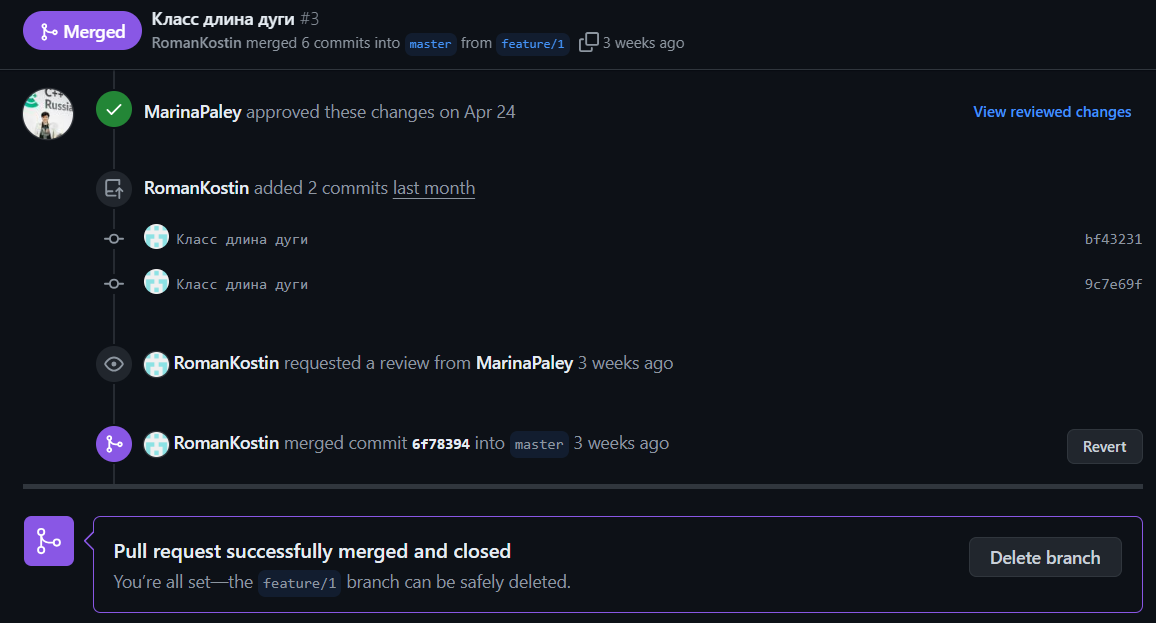


Рисунок 7 – Approve на класс Длина дуги