ЗВІТ

про виконання лабораторної роботи № 2.6

« Конструктори та перевантаження операцій для класів з вкладеними класами. Обчислення кількості об’єктів »

з дисципліни

«Об’єктно-орієнтоване програмування»

студента групи ІТ-12

М’ягких Владислава Юрійовича

**Мета роботи**

Освоїти використання конструкторів та перевантаження операцій для вкладених класів.

**Умова завдання:**

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

• конструктор ініціалізації (один або декілька),

• конструктор без аргументів і

• конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

• присвоєння,

• введення з клавіатури,

• виводу на екран,

• приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

• інкременту в обох формах (префіксній та постфіксній) і

• декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

Для демонстрації роботи з об’єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов’язково мають бути продемонстровані різні способи створення об’єктів і масивів об’єктів. Програма має демонструвати використання всіх функцій і методів. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

**Завдання наступне:**

Виконати завдання свого варіанту Лабораторної роботи № 1.6 (Вкладені класи) з конструкторами і перевантаженням операцій (тобто, Лабораторної роботи № 2.5), використовуючи конструкцію вкладеного класу замість композиції. Реалізувати обчислення кількості об’єктів внутрішнього та зовнішнього класів.

Метод Init() стане конструкторами, методи Read() та Display() – операціями вводу / виводу.

**Лабораторна робота № 1.6:**

Виконати завдання свого варіанту Лабораторної роботи № 1.5, використовуючи конструкцію вкладеного класу. Визначення класів та реалізації методів слід розмістити в окремих модулях.

**Лабораторна робота № 1.5:**

У всіх завданнях, крім вказаних в завданні операцій, обов’язково мають бути реалізовані наступні методи:

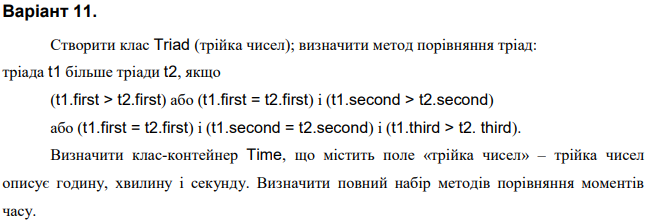
* методи доступу (константні методи зчитування та методи запису) значення кожного поля;
* метод ініціалізації Init( );
* метод введення з клавіатури Read( );
* метод виведення на екран Displау( );
* метод перетворення до літерного рядку toString( ).

Всі завдання мають бути реалізовані як класи із закритими полями, де операції реалізуються як методи класу.

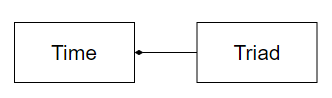
Визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об’єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов’язково мають бути продемонстровані різні способи створення об’єктів і масивів об’єктів. Програма має демонструвати використання всіх функцій і методів.

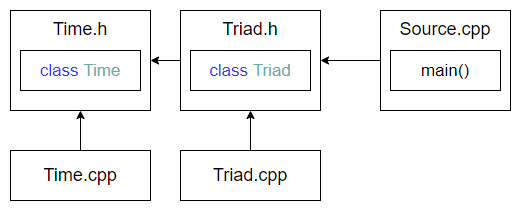
У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу- «контейнера», мають бути визначені як незалежні.



**UML-діаграма класів:**



**Структурна схема програми:**



**Текст програми:**

//////////////

// Source.cpp

//////////////

#include "Time.h"

using namespace std;

int main()

{

Time s1;

cout << "Time : " << Time::getCounter() << endl;

cout << "Time::Triad : " << Time::Triad::getCounter() << endl;

Time::Triad m1;

cout << "Time::Triad +1 : " << Time::Triad::getCounter() << endl;

{

Time s4(11, 19, 55);

cout << "Time local : " << Time::getCounter() << endl;

cout << "Time::Triad local : " << Time::Triad::getCounter() << endl;

Time::Triad m1;

cout << "Time::Triad local +1 : " << Time::Triad::getCounter() << endl;

}

cout << "Time : " << Time::getCounter() << endl;

cout << "Time::Triad : " << Time::Triad::getCounter() << endl;

Time::Triad t1, t2;

t1.Read();

t2.Read();

t1.Display();

cout << t2.toString() << endl;

if (t1.h1(t2))

{

cout << "t1 < t2" << endl;

}

else if (t1.m1(t2))

{

cout << "t1 = t2" << endl;

}

else

{

cout << "t1 > t2" << endl;

}

Time d1, d2;

d1.Init(t1);

d2.Init(t2);

d1.Display();

cout << d2.toString() << endl;

if (d1.h1(d2))

{

cout << "d1 < d2" << endl;

}

else if (d1.m1(d2))

{

cout << "d1 = d2" << endl;

}

else

{

cout << "d1 > d2" << endl;

}

return 0;

}

//////////////

// Time.h

//////////////

#pragma once

#include <string>

#include <iostream>

using namespace std;

class Time

{

private:

static int counter;

public:

class Triad

{

int first;

int second;

int third;

static int counter;

public:

int getFirst() const { return first; }

int getSecond() const { return second; }

int getThird() const { return third; }

bool setFirst(int);

bool setSecond(int);

bool setThird(int);

void Init(int firstValue, int secondValue, int thirdValue);

void Display() const;

void Read();

const string toString();

Triad();

Triad(const int first, const int second, const int third);

Triad(const Triad& m);

~Triad(void);

Triad& operator = (const Triad& m);

friend ostream& operator << (ostream& out, const Triad& m);

friend istream& operator >> (istream& in, Triad& m);

operator string () const;

Triad& operator ++ ();

Triad& operator -- ();

Triad operator ++ (int);

Triad operator -- (int);

bool h1(Triad t) const;

bool h2(Triad t) const;

bool m1(Triad t) const;

bool m2(Triad t) const;

bool s1(Triad t) const;

bool s2(Triad t) const;

double check();

static int getCounter();

};

Triad getTriad() const { return t; }

void setTriad(Triad tValue) { t = tValue; }

void Init(Triad t);

void Display() const;

void Read();

const string toString();

Time(const int first = 0, const int second = 0, const int third = 0);

Time(const Time& s);

~Time(void);

Time& operator = (const Time& s);

friend ostream& operator << (ostream& out, const Time& s);

friend istream& operator >> (istream& in, Time& s);

operator string () const;

Time& operator ++ ();

Time& operator -- ();

Time operator ++ (int);

Time operator -- (int);

bool h1(Time d) const;

bool h2(Time d) const;

bool m1(Time d) const;

bool m2(Time d) const;

bool s1(Time d) const;

bool s2(Time d) const;

static int getCounter();

private:

Triad t;

};

//////////////

// Time.cpp

//////////////

#include "Time.h"

#include <sstream>

using namespace std;

///////////////

/// class Time

int Time::counter = 0;

void Time::Init(Triad dValue)

{

setTriad(dValue);

}

void Time::Display() const

{

t.Display();

}

void Time::Read()

{

Triad tValue;

tValue.Read();

Init(tValue);

}

const string Time::toString()

{

stringstream sout;

sout << t.toString();

return sout.str();

}

Time::Time(const int first, const int second, const int third)

: t(first, second, third)

{

Time::counter++;

}

Time::Time(const Time& s)

{

t = s.t;

Time::counter++;

}

Time::~Time(void)

{

Time::counter--;

}

Time& Time::operator = (const Time& s)

{

t = s.t;

return \*this;

}

Time::operator string () const

{

stringstream ss;

ss << "t = " << t << endl;

return ss.str();

}

ostream& operator << (ostream& out, const Time& s)

{

out << string(s);

return out;

}

istream& operator >> (istream& in, Time& s)

{

cout << endl;

cout << "t = ? "; in >> s.t;

return in;

}

Time& Time::operator ++ ()

{

++t;

return\*this;

}

Time& Time::operator -- ()

{

--t;

return\*this;

}

Time Time::operator ++ (int)

{

Time s(\*this);

t++;

return\*this;

}

Time Time::operator -- (int)

{

Time s(\*this);

t--;

return\*this;

}

bool Time::h1(Time d) const

{

return t.h1(d.getTriad());

}

bool Time::h2(Time d) const

{

return t.h2(d.getTriad());

}

bool Time::m1(Time d) const

{

return t.m1(d.getTriad());

}

bool Time::m2(Time d) const

{

return t.m2(d.getTriad());

}

bool Time::s1(Time d) const

{

return t.s1(d.getTriad());

}

bool Time::s2(Time d) const

{

return t.s2(d.getTriad());

}

int Time::getCounter()

{

return Time::counter;

}

///////////////

/// class Triad

int Time::Triad::counter = 0;

bool Time::Triad::setFirst(int value)

{

if (value > -1 && value < 24)

{

first = value;

return true;

}

else

{

return false;

}

}

bool Time::Triad::setSecond(int value)

{

if (value > -1 && value < 60)

{

second = value;

return true;

}

else

{

return false;

}

}

bool Time::Triad::setThird(int value)

{

if (value > -1 && value < 60)

{

third = value;

return true;

}

else

{

return false;

}

}

void Time::Triad::Init(int firstValue, int secondValue, int thirdValue)

{

setFirst(firstValue);

setSecond(secondValue);

setThird(thirdValue);

}

void Time::Triad::Display() const

{

cout << first << "." << second << "." << third << endl;

}

void Time::Triad::Read()

{

int firstValue;

int secondValue;

int thirdValue;

cout << "Enter triad " << endl << endl;

cout << "first = "; cin >> firstValue;

cout << "second = "; cin >> secondValue;

cout << "third = "; cin >> thirdValue;

Init(firstValue, secondValue, thirdValue);

}

const string Time::Triad::toString()

{

stringstream sout;

sout << first << "." << second << "." << third;

return sout.str();

}

Time::Triad::Triad()

: first(0), second(0), third(0)

{

Time::Triad::counter++;

}

Time::Triad::Triad(const int first, const int second, const int third)

: first(first), second(second), third(third)

{

Time::Triad::counter++;

}

Time::Triad::Triad(const Triad& m)

: first(m.first), second(m.second), third(m.third)

{

Time::Triad::counter++;

}

Time::Triad::~Triad(void)

{

Time::Triad::counter--;

}

Time::Triad& Time::Triad::operator = (const Time::Triad& m)

{

this->first = m.first;

this->second = m.second;

this->third = m.third;

return \*this;

}

Time::Triad::operator string () const

{

stringstream ss;

ss << endl;

ss << "first = " << first << endl;

ss << "second = " << second << endl;

ss << "third = " << third << endl;

return ss.str();

}

ostream& operator << (ostream& out, const Time::Triad& m)

{

out << string(m);

return out;

}

istream& operator >> (istream& in, Time::Triad& m)

{

int first;

int second;

int third;

cout << endl;

cout << "first = ? "; in >> first;

cout << "second = ? "; in >> second;

cout << "third = ? "; in >> third;

m.setFirst(first);

m.setSecond(second);

m.setThird(third);

return in;

}

Time::Triad& Time::Triad::operator ++ ()

{

++first;

++second;

++third;

return \*this;

}

Time::Triad& Time::Triad::operator -- ()

{

--first;

--second;

--third;

return \*this;

}

Time::Triad Time::Triad::operator ++ (int)

{

Triad t(\*this);

first++;

second++;

third++;

return t;

}

Time::Triad Time::Triad::operator -- (int)

{

Triad t(\*this);

first--;

second--;

third--;

return t;

}

bool Time::Triad::h1(Triad t) const

{

if (first < t.first)

{

return true;

}

if (first > t.first)

{

return false;

}

if (second < t.second)

{

return true;

}

if (second > t.second)

{

return false;

}

if (third < t.third)

{

return true;

}

if (third > t.third)

{

return false;

}

return false;

}

bool Time::Triad::h2(Triad t) const

{

if (first < t.first)

{

return true;

}

if (first > t.first)

{

return false;

}

if (second < t.second)

{

return true;

}

if (second > t.second)

{

return false;

}

if (third < t.third)

{

return true;

}

if (third > t.third)

{

return false;

}

return true;

}

bool Time::Triad::m1(Triad t) const

{

if ((first == t.first) && (second == t.second) && (third == t.third))

{

return true;

}

return false;

}

bool Time::Triad::m2(Triad t) const

{

if ((first == t.first) && (second == t.second) && (third == t.third))

{

return false;

}

return true;

}

bool Time::Triad::s1(Triad t) const

{

if (first < t.first)

{

return false;

}

if (first > t.first)

{

return true;

}

if (second < t.second)

{

return false;

}

if (second > t.second)

{

return true;

}

if (third < t.third)

{

return false;

}

if (third > t.third)

{

return true;

}

return false;

}

bool Time::Triad::s2(Triad t) const

{

if (first < t.first)

{

return false;

}

if (first > t.first)

{

return true;

}

if (second < t.second)

{

return false;

}

if (second > t.second)

{

return true;

}

if (third < t.third)

{

return false;

}

if (third > t.third)

{

return true;

}

return true;

}

double Time::Triad::check()

{

return first > second;

}

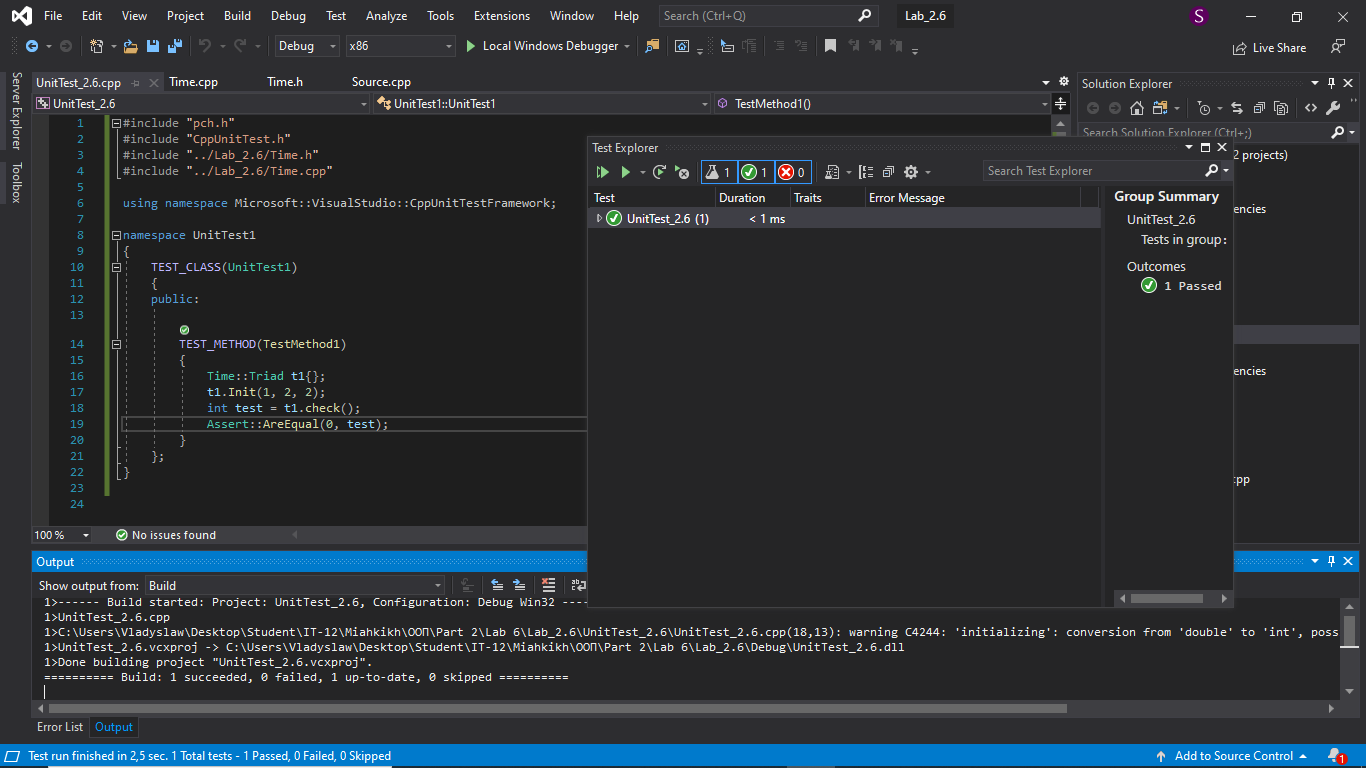
int Time::Triad::getCounter()

{

return Time::Triad::counter;

}

**Результати unit-тесту:**



**Посилання на git-репозиторій з проектом:**

**Висновки:**

На даній лабораторній роботі, я освоїв використання конструкторів та перевантаження операцій для вкладених класів.