ЗВІТ

про виконання лабораторної роботи № 2.5

« Конструктори та перевантаження операцій для класів з композицією »

з дисципліни

«Об’єктно-орієнтоване програмування»

студента групи ІТ-12

М’ягких Владислава Юрійовича

**Мета роботи**

Освоїти використання конструкторів та перевантаження операцій для класів з композицією.

**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 № 1.5 (Композиція класів та об’єктів) з конструкторами і перевантаженням операцій.

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

**Лабораторна робота № 1.5:**

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

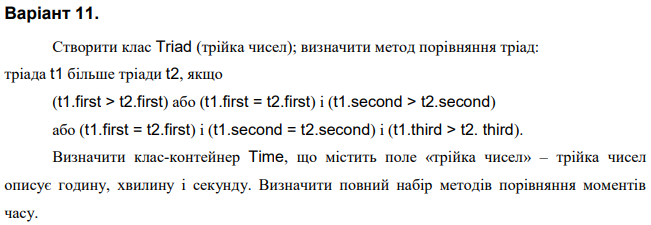
* методи доступу (константні методи зчитування та методи запису) значення кожного поля;
* метод ініціалізації Init( );
* метод введення з клавіатури Read( );
* метод виведення на екран Displау( );
* метод перетворення до літерного рядку toString( ).

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

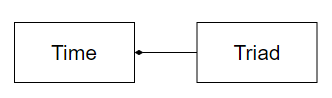
Визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

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.

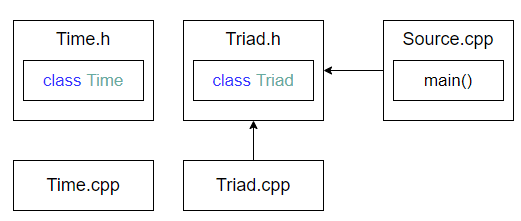
У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу- «контейнера», мають бути визначені як незалежні.



**UML-діаграма класів:**



**Структурна схема програми:**



**Текст програми:**

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

// Source.cpp

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

#include "Time.h"

#include <iostream>

using namespace std;

int main()

{

Time s1;

Time s4(11, 19, 55);

cout << s4 << endl;

s1 = ++s4;

cout << s1 << endl;

cout << s4 << endl;

s1 = --s4;

cout << s1 << endl;

cout << s4 << endl;

s1 = s4++;

cout << s1 << endl;

cout << s4 << endl;

s1 = s4--;

cout << s1 << endl;

cout << s4 << endl;

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;

}

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

// Triad.h

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

#pragma once

#include <string>

#include <sstream>

using namespace std;

class Triad

{

int first;

int second;

int third;

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& 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);

~Triad(void);

friend bool G(const Triad& t1, const Triad& t2);

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();

};

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

// Triad.cpp

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

#include "Triad.h"

#include <iostream>

#include <sstream>

using namespace std;

bool Triad::setFirst(int value)

{

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

{

first = value;

return true;

}

else

{

return false;

}

}

bool Triad::setSecond(int value)

{

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

{

second = value;

return true;

}

else

{

return false;

}

}

bool Triad::setThird(int value)

{

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

{

third = value;

return true;

}

else

{

return false;

}

}

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

{

setFirst(firstValue);

setSecond(secondValue);

setThird(thirdValue);

}

void Triad::Display() const

{

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

}

void 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 Triad::toString()

{

stringstream sout;

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

return sout.str();

}

Triad::Triad()

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

{}

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

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

{}

Triad::Triad(const Triad& m)

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

{}

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

{

this->first = m.first;

this->second = m.second;

this->third = m.third;

return \*this;

}

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

{

out << string(m);

return out;

}

istream& operator >> (istream& in, 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;

}

Triad::operator string () const

{

stringstream ss;

ss << endl;

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

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

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

return ss.str();

}

Triad& Triad::operator ++ ()

{

++first;

++second;

++third;

return \*this;

}

Triad& Triad::operator -- ()

{

--first;

--second;

--third;

return \*this;

}

Triad Triad::operator ++ (int)

{

Triad t(\*this);

first++;

second--;

third--;

return t;

}

Triad Triad::operator -- (int)

{

Triad t(\*this);

first--;

second--;

third--;

return t;

}

Triad::~Triad(void)

{}

bool G(const Triad& t1, const Triad& t2)

{

return (t1.first > t2.first) || (t1.first == t2.first && t1.second > t2.second) || (t1.first == t2.first && t1.second == t2.second && t1.third > t2.third);

}

bool 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 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 Triad::m1(Triad t) const

{

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

{

return true;

}

return false;

}

bool Triad::m2(Triad t) const

{

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

{

return false;

}

return true;

}

bool 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 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 Triad::check()

{

return first > second;

}

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

// Time.h

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

#pragma once

#include <string>

#include "Triad.h"

using namespace std;

class Time {

Triad t;

public:

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& 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);

~Time(void);

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;

};

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

// Time.cpp

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

#include "Time.h"

#include <iostream>

#include <sstream>

using namespace std;

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::Time(const Time& s)

{

t = s.t;

}

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

{

t = s.t;

return \*this;

}

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::operator string () const

{

stringstream ss;

return string(t) + ss.str();

}

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;

}

Time::~Time(void)

{}

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());

}

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

**Висновки:**

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