МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №2**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Пономарев Никита Владимирович, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

**Задание:** Разработать программу на языке C++ согласно варианту задания. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод. Реализовать пользовательский литерал для работы с константами объектов созданного класса.

**Вариант №19:**

Создать класс Address для работы с адресами домов. Адрес должен состоять из строк с названием города и улицы и чисел с номером дома и квартиры. Реализовать операции сравнения адресов, а также операции проверки принадлежности адреса к улице и городу. В операциях не должен учитываться регистр строки. Так же необходимо сделать операцию, которая возвращает истину если два адреса находятся по соседству (на одной улице в одном городе и дома стоят подряд).

**Описание программы:**

Исходный код разделён на 3 файла:

* adress.h – описание класса адрес
* adress.cpp – реализация класса адрес
* main.cpp – основная программа

**Дневник отладки:**

Проблем не возникло.

**Вывод:**  
 В процессе выполнения работы я на практике познакомился с пользовательскими литералами. Это очень удобная и практическая вещь, о которой я не знал до курса ООП. Использование этого средства позволяет получать из заданных типов данных какие то данные, вычислять что то, без использования функций, а с помощью переопределения специального оператора

**Исходный код:**

**adress.h:**

**#ifndef ADRESS\_H**

#define ADRESS\_H

#include <iostream>

using namespace std;

class Adress {

public:

Adress() = default;

Adress(string c, string r, int h, int a): city(c), route(r), house\_number(h), apartaments\_number(a){}

friend ostream& operator<<(ostream& s, const Adress& l);

friend Adress operator+(const Adress& l, const Adress& r);

bool operator==(const Adress& l) const;

bool is\_near(const Adress& l) const;

bool adress\_to\_route(string route\_) const;

bool adress\_to\_city(string city\_) const;

string Get\_city() const;

string Get\_route() const;

int Get\_house\_number() const;

int Get\_apartaments\_number() const;

private:

string city;

string route;

int house\_number;

int apartaments\_number;

};

*/\*Создать класс Address для работы с адресами домов. Адрес должен состоять из строк с названием города*

*и улицы и чисел с номером дома и квартиры. Реализовать операции сравнения адресов, а также операции*

*проверки принадлежности адреса к улице и городу. В операциях не должен учитываться регистр строки.*

*Так же необходимо сделать операцию, которая возвращает истину если два адреса находятся по соседству*

*(на одной улице в одном городе и дома стоят подряд).\*/*

#endif

**adress.cpp:**

#include "adress.h"

string Adress::Get\_city() const {

return city;

}

string Adress::Get\_route() const {

return route;

}

int Adress::Get\_house\_number() const {

return house\_number;

}

int Adress::Get\_apartaments\_number() const {

return apartaments\_number;

}

ostream& operator<<(ostream& s, const Adress& l){

s << "(" << l.Get\_city() << ", " << l.Get\_route() << ", " << l.Get\_house\_number() << ", " << l.Get\_apartaments\_number() << ")";

return s;

}

Adress operator+(const Adress& l, const Adress& r){

Adress q;

string city = "";

string route = "";

int house\_number = 0;

int apartaments\_number = 0;

if(l.Get\_city() != ""){

city = l.Get\_city();

}

if (l.Get\_route() != ""){

route = l.Get\_route();

}

if (l.Get\_house\_number()){

house\_number = l.Get\_house\_number();

}

if (l.Get\_apartaments\_number()){

apartaments\_number = l.Get\_apartaments\_number();

}

if(r.Get\_city() != ""){

city = r.Get\_city();

}

if (r.Get\_route() != ""){

route = r.Get\_route();

}

if (r.Get\_house\_number()){

house\_number = r.Get\_house\_number();

}

if (r.Get\_apartaments\_number()){

apartaments\_number = r.Get\_apartaments\_number();

}

return Adress(city, route, house\_number, apartaments\_number);

}

bool Adress::operator==(const Adress& l) const {

return l.city == city && l.route == route && l.house\_number == house\_number && l.apartaments\_number == apartaments\_number;

}

bool Adress::is\_near(const Adress& l) const {

if(l.Get\_city() == city && l.Get\_route() == route){

return (abs(l.Get\_apartaments\_number() - apartaments\_number) <= 1) || (abs(l.Get\_house\_number() - house\_number) <= 1);

} else {

return false;

}

}

bool Adress::adress\_to\_route(string route\_) const {

return route == route\_;

}

bool Adress::adress\_to\_city(const string city\_) const {

return city == city\_;

}

**main.cpp:**

**#include <iostream>**

#include "adress.h"

using namespace std;

Adress operator""\_city(const char\* s, std::size\_t n){

string city = "";

for(int i = 0; i < n; ++i){

city += s[i];

}

return Adress(city, "", 0, 0);

}

Adress operator""\_route(const char\* r, std::size\_t n){

string route = "";

for(int i = 0; i < n; ++i){

route += r[i];

}

return Adress("", route, 0, 0);

}

Adress operator""\_house\_number(unsigned long long int house\_number){

return Adress("", "", house\_number, 0);

}

Adress operator""\_apartaments\_number(unsigned long long int apartaments\_number){

return Adress("", "", 0, apartaments\_number);

}

int main(){

Adress a("Moscow", "Tverskaya", 4, 5);

Adress b("Moscow", "Tverskaya", 5, 100);

Adress c("Moscow", "Petrovskaya", 13, 56);

Adress d("Moscow", "Tverskaya", 4, 5);

cout << a << "**\n**" << b.is\_near(a) << "**\n**" << (a == b) << "**\n**" << (a == d) << "**\n**" << b.adress\_to\_route("Tverskaya") << "**\n**";

cout << "Mосква"\_city + "проспект 60-летия Октября"\_route + 9\_house\_number + 12\_apartaments\_number << "**\n**";

}

**Пример работы:**

(Moscow, Tverskaya, 4, 5)

1

0

1

1

(Mосква, проспект 60-летия Октября, 9, 12)