МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

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

Преподаватель *Дорохов Евгений Павлович*

Условие

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).
Реализовать над объектами реализовать в виде перегрузки операторов. Реализовать пользовательский литерал для работы с константами объектов созданного класс Исходный код лежит в 3 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством команд из меню
2. adress.h: описание класса адресов
3. adress.cpp: реализация класса адреса

Протокол работы

(Moscow, Tverskaya, 4, 5)
1
0
1
1

Дневник отладки

Проблем и ошибок при написании данной работы не возникло.

Недочёты

Выводы

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), ap
        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 на
 и улицы и чисел с номером дома и квартиры. Реализовать операции сравнения адресов, а та
 проверки принадлежности адреса к улице и городу. В операциях не должен учитываться реги
 Так же необходимо сделать операцию, которая возвращает истину если два адреса находятся
  (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() <<
    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.apa
}

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
    } 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" <<
    cout << "Москва"_city + "проспект 60-летия Октября"_route + 9_house_number + 12_apar
}
```