

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

# ЛАБОРАТОРНАЯ РАБОТА №1

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

Студент Прохоров Данила Михайлович, группа М80-208Б-20  
Преподаватель Дорохов Евгений Павлович

## Цель:

- Изучение системы сборки на языке C++, изучение систем контроля версии.
- Изучение основ работы с классами в C++;

## **Порядок выполнения работы**

1. Ознакомиться с теоретическим материалом.
2. Получить у преподавателя вариант задания.
3. Реализовать задание своего варианта в соответствии с поставленными требованиями.
4. Подготовить тестовые наборы данных.
5. Создать репозиторий на GitHub.
6. Отправить файлы лабораторной работы в репозиторий.
7. Отчитаться по выполненной работе путём демонстрации работающей программы на тестовых наборах данных (как подготовленных самостоятельно, так и предложенных преподавателем) и ответов на вопросы преподавателя (как из числа контрольных, так и по реализации программы).

## **Требования к программе**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться **oop\_exercise\_01** (в случае использования Windows **oop\_exercise\_01.exe**)

Необходимо зарегистрироваться на GitHub (если студент уже имеет регистрацию на GitHub то можно использовать ее) и создать репозиторий для задания лабораторной работы.

Преподавателю необходимо предъявить ссылку на публичный репозиторий на Github. Имя репозитория должно быть [https://github.com/login/oop\\_exercise\\_01](https://github.com/login/oop_exercise_01)

Где login – логин, выбранный студентом для своего репозитория на Github.

Репозиторий должен содержать файлы:

- `main.cpp` // файл с заданием работы
- `CMakeLists.txt` // файл с конфигураций CMake
- `test_xx.txt` // файл с тестовыми данными. Где xx – номер тестового набора 01, 02 , ... Тестовых наборов должно быть больше 1.
- `report.doc` // отчет о лабораторной работе

Вариант:

- |    |  |
|----|--|
| 19 | Создать класс <b>Address</b> 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 лежит в 3 файлах:

1. `main.cpp` - исполняемый код.
2. `address.h` - специальный файл `.h`, содержащий прототипы используемых мною функций.
3. `address.cpp` - реализация функций для моего задания.
4. `CMakeLists.txt` - специальный дополнительный файл типа CMakeLists.

## Дневник отладки

С первого раза получилось сделать работающую программу, благо задание было несложным.

## Недочёты

Недочёты не обнаружил.

## Выводы

Данная лабораторная работа помогла мне использовать полученные на лекциях теоретические знания на практике, и я написал простой класс, попрактиковавшись вдобавок в перегрузке операторов.

## Исходный код

### address.h

```
#include <cmath>
#include <string>
#include <iostream>
#include <limits>
class Address
{
private:
    std::string city, street;
    unsigned house, flat;
public:
    Address(std::string, std::string, unsigned, unsigned);
    Address(std::istream&);
    void operator= (const Address&);
    bool operator== (const Address&);
    bool belonding(std::string, std::string);
    bool neighbours(Address);
    void set_new_address(std::string, std::string, unsigned, unsigned);
    friend std::ostream& operator <<(std::ostream& os, const Address& address);
    friend std::istream& operator >> (std::istream&, Address&);
    ~Address();
};

bool equals(std::string, std::string);

#endif
```

### address.cpp

```
#include "address.h"

bool equals(std::string s1, std::string s2)
{
    if (s1.length() == s2.length())
    {
        for (int i = 0; i < s1.length(); i++)
        {
```

```

        if (tolower(s1[i]) != tolower(s2[i]))
        {
            return false;
        }
    }
    return true;
}
return false;
}

Address::Address(std::string city, std::string street, unsigned house, unsigned flat):
city(city), street(street), house(house), flat(flat)
{
}

Address::Address(std::istream& is)
{
    std::string aux, h, f;
    //is.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    getline(is, aux);
    if (aux == "")
    {
        getline(is, city);
    }
    else
    {
        city = aux;
    }
    //getline(is, city);
    getline(is, street);
    getline(is, h);
    getline(is, f);
    this->house = unsigned(stoi(h));
    this->flat = unsigned(stoi(f));
}

void Address::operator= (const Address& address)
{
    city = address.city;
    street = address.street;
    house = address.house;
    flat = address.flat;
}

bool Address::operator== (const Address& address)
{
    std::string adcity = address.city, adstreet = address.street;
    if (equals(city, address.city) && (equals(street, address.street)) && (house ==
address.house) && (flat == address.flat))
    {
        return true;
    }
    return false;
}

bool Address::belonding(std::string city, std::string street)
{
    if (equals(this->city, city) && (equals(this->street, street)))

```

```

        {
            return true;
        }
        return false;
    }

bool Address::neighbours(Address address)
{
    if (equals(city, address.city) && (equals(street, address.street)))
    {
        return true;
    }
    return false;
}

void Address::set_new_address(std::string city, std::string street, unsigned house,
unsigned flat)
{
    this->city = city;
    this->street = street;
    this->house = house;
    this->flat = flat;
}

std::istream& operator>> (std::istream& is, Address& address)
{
    std::string aux, h, f;
    //is.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    getline(is, aux);
    if (aux == "\n")
    {
        getline(is, address.city);
    }
    else
    {
        address.city = aux;
    }
    //getline(is, address.city);
    getline(is, address.street);
    getline(is, h);
    getline(is, f);
    address.house = unsigned(stoi(h));
    address.flat = unsigned(stoi(f));
    return is;
}

std::ostream& operator << (std::ostream& os, const Address& address)
{
    os << "City: " << address.city << "\n";
    os << "Street: " << address.street << "\n";
    os << "House: " << address.house << "\n";
    os << "Flat: " << address.flat;
    return os;
}

Address::~Address()
{
}

```

## main.cpp

```
#include "address.h"

int main()
{
    std::string city1 = "Moscow", street1 = "Arbat";
    unsigned house1 = 5, flat1 = 24;
    Address address1(city1, street1, house1, flat1);
    std::cout << address1.belonding("Moscow", "Neglinnaya") << "\n";
    std::cout << address1.belonding("Moscow", "Arbat") << "\n";
    std::cout << address1 << "\n";
    Address address156(std::cin);
    std::cout << address156 << "\n";
    address156 = Address(std::cin);
    std::cout << address156 << "\n";
    system("pause");
    return 0;
}
```