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 *Прохоров Данила Михайлович, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

**Цель:**

● Изучение основ работы с классами в С++;

● Перегрузка операций и создание литералов

## 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.

Реализовать над объектами реализовать в виде перегрузки операторов.

Реализовать пользовательский литерал для работы с константами объектов созданного класса.

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**

Программа в отладке не нуждалась, всё работало так, как и было задумано.

**Недочёты**  
Недочёты обнаружены не были.

**Выводы**

Лабораторная работа №2 – усовершенствованная лабораторная работа №1, в которую были добавлены пользовательские литералы и перегрузка операторов. Лабораторная была выполнена успешно.

**Исходный код**

address.h

#ifndef ADDRESS\_H

#define ADDRESS\_H

#include <cmath>

#include <string>

#include <iostream>

#include <limits>

#include <vector>

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::vector<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);

std::string rem(std::string);

Address operator "" \_address(const char\* str, size\_t size);

std::vector<std::string> operator"" \_city\_and\_street(const char\* str, size\_t size);

#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;

}

std::string rem(std::string string)

{

while (string[0] == ' ')

{

string.erase(0, 1);

}

while (string[string.size() - 1] == ' ')

{

string.erase(string.size() - 1);

}

return string;

}

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::vector<std::string> strings)

{

if (equals(this->city, strings[0]) && (equals(this->street, strings[1])))

{

return true;

}

return false;

}

bool Address::neighbours(Address address)

{

if (equals(city, address.city) && (equals(street, address.street)) && (((house == address.house + 1) && (address.house != UINT\_MAX)) ||

((address.house == house + 1) && (house != UINT\_MAX))))

{

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()

{

}

Address operator "" \_address(const char\* str, size\_t size)

{

std::string string = std::string(str, size);

std::vector<std::string> parts;

size\_t n;

do

{

n = string.find(",");

if (n != std::string::npos)

{

parts.push\_back(rem(string.substr(0, n)));

string.erase(0, n + 1);

}

if (parts.size() == 4)

{

break;

}

} while (n != std::string::npos);

parts.push\_back(rem(string));

if (parts.size() < 4)

{

return Address("", "", 0, 0);

}

return Address(parts[0], parts[1], unsigned(stoi(parts[2])), unsigned(stoi(parts[3])));

}

std::vector<std::string> operator "" \_city\_and\_street(const char\* str, size\_t size)

{

std::string string = std::string(str, size);

std::vector<std::string> parts;

size\_t n;

do

{

n = string.find(",");

if (n != std::string::npos)

{

parts.push\_back(rem(string.substr(0, n)));

string.erase(0, n + 1);

}

if (parts.size() == 2)

{

break;

}

} while (n != std::string::npos);

parts.push\_back(rem(string));

if (parts.size() < 2)

{

return {"", ""};

}

return {parts[0], parts[1]};

}

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::string city2 = "Moscow", street2 = "Arbat";

unsigned house2 = 4, flat2 = 19;

Address address2(city2, street2, house2, flat2);

std::cout << address1.neighbours(address2) << "\n";

std::cout << address1.belonding({"Moscow", "Neglinnaya"}) << "\n";

std::cout << address1.belonding({"Moscow", "Arbat"}) << "\n";

std::cout << address1 << "\n";

std::string city3 = "Moscow", street3 = "Olega Tsareva";

unsigned house3 = 12, flat3 = 2;

Address address3(city3, street3, house3, flat3);

std::cout << address2 << "\n";

std::cout << address2 << "\n";

Address address156(std::cin);

std::cout << address156 << "\n";

address156 = Address(std::cin);

std::cout << address156 << "\n";

std::cout << address1.neighbours(address3);

Address address4(std::cin);

std::cout << address4.neighbours(Address(std::cin)) << "\n";

std::cout << address1.neighbours("Moscow, Neglinnaya, 19, 89"\_address) << "\n";

std::cout << address1.neighbours("Moscow, Arbat, 6, 4"\_address) << "\n";

std::cout << address2.belonding("Moscow, Olega Tsareva"\_city\_and\_street) << "\n";

std::cout << address2.belonding("Moscow, Arbat"\_city\_and\_street) << "\n";

system("pause");

return 0;

}