

**Московский авиационный институт
(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 1

Тема: Простые классы на языке C++

Студент: Короткевич Леонид
Витальевич

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

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).

2. Описание программы

Address.hpp	
std::string to_lower(const std::string &str)	функция для сравнения строк без учета регистра
Address()	конструктор по умолчанию
Address(const std::string &new_town, const std::string &new_street, const int &new_aprt, const int &new_house)	конструктор с 4 параметрами: новые город, улица, дом, квартира
bool operator==(Address const &rhs) const	оператор сравнения экземпляров класса
bool inTown(std::string const &aTown) const	константный метод: принадлежность городу
bool inStreet(std::string const &aStreet) const	константный метод: принадлежность улице
bool isNeighs(Address const &rhs) const	константный метод: проверка, является ли соседом
void display() const	константный метод: вывод инф. об адресе
Vector.hpp	
Vector();	конструктор по умолчанию
Vector(const int &n);	конструктор с 1 параметром: размер вектора
Vector(const int &n, T elem);	конструктор с 2 параметрами: размер вектора, заполнить элементами T
~Vector();	деструктор по умолчанию
void assert(const int &n, T elem);	метод: заполнить вектор n одинаковыми элементами
T &operator[](const int &index) const;	конст. метод: обратиться к элементу вектора
void push_back(T elem);	метод: добавить в конец вектора элемент
unsigned int size() const;	конст. метод: получить размер вектора
void erase(const int &idx);	метод: удалить элемент из вектора

3. Набор тестов

test_01.txt	test_02.txt	test_03.txt
1	1	1
msk fest 4 3	ekb lenina 123 1	ufa zelenka 2 3
1	1	1
spb fest 4 3	EKB LENINa 123 1	ufa ZELENKA 4 5
1	1	1
msK fesT 4 3	ekb lenina 123 1	istra lenina 2 1
3	7	4
1 2	6	1 zelenKA
3	1 3	4
1 3	6	1 zelenka
7	1 2	4
2	1	2 ZELENKa
2	EKB LENINA 124 0	5
7	7	1 UFA
8	6	5
	1 4	3 IsTrA
	3	6
	1 4	1 2
	8	8

4. Результаты выполнения тестов

test_01.txt	test_02.txt	test_03.txt																																								
<div>1 - добавить адрес</div> <div>2 - удалить адрес</div> <div>3 - сравнить два адреса</div> <div>4 - проверить принадлежность улице</div> <div>5 - проверить принадлежность городу</div> <div>6 - являются ли соседями</div> <div>7 - вывести список адресов</div> <div>8 - выйти</div> <div>Адрес msk fest 4 3 успешно добавлен в список</div> <div>Адрес spb fest 4 3 успешно добавлен в список</div> <div>Адрес msK fesT 4 3 успешно добавлен в список</div> <div>Адреса НЕ равны</div> <div>Адреса равны</div> <table><tr><th>№</th><th>Город</th><th>Улица</th><th>Дом</th></tr><tr><td>1</td><td>msk</td><td>fest</td><td>3</td></tr><tr><td>2</td><td>spb</td><td>fest</td><td>3</td></tr><tr><td>3</td><td>msK</td><td>fesT</td><td>3</td></tr><tr><td>4</td><td></td><td></td><td></td></tr></table> <div>Адрес spb fest 3 4 успешно</div>	№	Город	Улица	Дом	1	msk	fest	3	2	spb	fest	3	3	msK	fesT	3	4				<div>1 - добавить адрес</div> <div>2 - удалить адрес</div> <div>3 - сравнить два адреса</div> <div>4 - проверить принадлежность улице</div> <div>5 - проверить принадлежность городу</div> <div>6 - являются ли соседями</div> <div>7 - вывести список адресов</div> <div>8 - выйти</div> <div>Адрес ekb lenina 123 1 успешно добавлен в список</div> <div>Адрес EKB LENINa 123 1 успешно добавлен в список</div> <div>Адрес ekb lenina 123 1 успешно добавлен в список</div> <table><tr><th>№</th><th>Город</th><th>Улица</th><th>Дом</th></tr><tr><td>1</td><td>ekb</td><td>lenina</td><td>1</td></tr><tr><td>2</td><td>EKB</td><td>LENINa</td><td>1</td></tr><tr><td>3</td><td>ekb</td><td>lenina</td><td>1</td></tr><tr><td>4</td><td></td><td></td><td></td></tr></table>	№	Город	Улица	Дом	1	ekb	lenina	1	2	EKB	LENINa	1	3	ekb	lenina	1	4				<div>1 - добавить адрес</div> <div>2 - удалить адрес</div> <div>3 - сравнить два адреса</div> <div>4 - проверить принадлежность улице</div> <div>5 - проверить принадлежность городу</div> <div>6 - являются ли соседями</div> <div>7 - вывести список адресов</div> <div>8 - выйти</div> <div>Адрес ufa zelenka 2 3 успешно добавлен в список</div> <div>Адрес ufa ZELENKA 4 5 успешно добавлен в список</div> <div>Адрес istra lenina 2 1 успешно добавлен в список</div> <div>Адрес под номером 1 действительно принадлежит улице zelenKA</div> <div>Адрес под номером 1 действительно принадлежит улице zelenka</div> <div>Адрес под номером 2 действительно принадлежит улице ZELENKa</div> <div>Адрес под номером 1 действительно принадлежит городу UFA</div> <div>Адрес под номером 3 действительно принадлежит городу IsTrA</div>
№	Город	Улица	Дом																																							
1	msk	fest	3																																							
2	spb	fest	3																																							
3	msK	fesT	3																																							
4																																										
№	Город	Улица	Дом																																							
1	ekb	lenina	1																																							
2	EKB	LENINa	1																																							
3	ekb	lenina	1																																							
4																																										

удален из списка				Адреса совпадают				Адреса под номерами 1 и 2 НЕ являются соседними			
№	Город	Улица	Дом	Адреса совпадают							
1	Квартира msk	fest	3	Адрес EKB LENINA 124 0 успешно добавлен в список							
2	4 msK	4 fesT	3	№	Город	Улица	Дом				
					Квартира						
				1	ekb	lenina	1				
					123						
				2	EKB	LENINa	1				
					123						
				3	ekb	lenina	1				
					123						
				4	EKB	LENINA	0				
					124						
				Адреса под номерами 1 и 4 являются соседними							
				Адреса НЕ равны							

5. Листинг программы

main.cpp:

```

/*
Короткевич Л. В.
github.com/anxieuse
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).
*/

#include <iostream>
#include <string>

#include "Address.hpp"
#include "Vector.hpp"

int main()
{
    std::string town, street;
    int apt, house, idToDel, id, id1, id2;
    Vector<Address> ads;

    int command_idx;
    bool input = true;
    std::cout << '\n'
        << " 1 - добавить адрес" << '\n'
        << " 2 - удалить адрес" << '\n'
        << " 3 - сравнить два адреса" << '\n'

```

```

    << " 4 - проверить принадлежность улице" << '\n'
    << " 5 - проверить принадлежность городу" << '\n'
    << " 6 - являются ли соседями" << '\n'
    << " 7 - вывести список адресов" << '\n'
    << " 8 - выйти\n"
    << '\n';
while (input)
{
    std::cin >> command_idx;
    switch (command_idx)
    {
        case 1:
        {
            std::cin >> town >> street >> apt >> house;
            printf("Адрес %s %s %d %d успешно добавлен в список\n", town.c_str(), street.c_str(),
apt, house);
            ads.push_back({town, street, apt, house});
            break;
        }
        case 2:
        {
            std::cin >> idToDel;
            --idToDel;
            if (idToDel < ads.size() && idToDel >= 0)
            {
                std::cout << "Адрес ";
                ads[idToDel].display();
                std::cout << " успешно удален из списка\n";
                ads.erase(idToDel);
            }
            else
            {
                std::cout << "Введен некорректный индекс\n";
            }
            break;
        }
        case 3:
        {
            std::cin >> id1 >> id2;
            if (ads[id1 - 1] == ads[id2 - 1])
            {
                std::cout << "Адреса равны\n";
            }
            else
            {
                std::cout << "Адреса НЕ равны\n";
            }
            break;
        }
        case 4:
        {
            std::cin >> id >> street;
            if (ads[id - 1].inStreet(street))

```

```

        {
            printf("Адрес под номером %d действительно принадлежит улице %s\n", id,
street.c_str());
        }
        else
        {
            printf("Адрес под номером %d НЕ принадлежит улице %s\n", id, street.c_str());
        }
        break;
    }
    case 5:
    {
        std::cin >> id >> town;
        if (ads[id - 1].inTown(town))
        {
            printf("Адрес под номером %d действительно принадлежит городу %s\n", id,
town.c_str());
        }
        else
        {
            printf("Адрес под номером %d НЕ принадлежит городу %s\n", id, town.c_str());
        }
        break;
    }
    case 6:
    {
        std::cin >> id1 >> id2;
        if (ads[id1 - 1].Address::isNeighs(ads[id2 - 1]))
        {
            if (ads[id1 - 1] == ads[id2 - 1])
                printf("Адреса совпадают\n");
            else
                printf("Адреса под номерами %d и %d являются соседними\n", id1, id2);
        }
        else
        {
            printf("Адреса под номерами %d и %d НЕ являются соседними\n", id1, id2);
        }
        break;
    }
    case 7:
    {
        if (ads.size() == 0)
        {
            std::cout << "Список адресов пуст\n";
        }
        else
        {
            printf("\n№\tГород\tУлица\tДом\tКвартира\n");
            for (int i = 0; i < ads.size(); ++i)
            {
                std::cout << i + 1 << '\t';
                // std::cout << &x - &ads[0] + 1 << '\t';
            }
        }
    }
}

```

```

        ads[i].Address::display();
        std::cout << '\n';
    }
    std::cout << '\n';
}
break;
}
case 8:
{
    input = false;
    break;
}
}
}
}

```

Vector.hpp:

```

#ifndef VECTOR_HPP
#define VECTOR_HPP

template <class T>
class Vector
{
public:
    Vector();
    Vector(const int &n);
    Vector(const int &n, T elem);
    ~Vector();

    void assert(const int &n, T elem);
    T &operator[](const int &index) const;
    void push_back(T elem);
    unsigned int size() const;
    void erase(const int &idx);

private:
    unsigned int capacity;
    unsigned int maxsize;
    T *data;
};

template <class T>
void Vector<T>::assert(const int &n, T elem)
{
    for (int i = 0; i < n; i++)
    {
        data[i] = elem;
    }
}

template <class T>

```

```

T &Vector<T>::operator[](const int &index) const
{
    return data[index];
}

```

```

template <class T>
void Vector<T>::push_back(T elem)
{
    if (data == 0)
    {
        maxsize = 1;
        data = new T[maxsize];
    }
    if (capacity == maxsize)
    {
        maxsize = maxsize * 2;
        T *new_data = new T[maxsize];
        for (int i = 0; i < capacity; i++)
        {
            new_data[i] = data[i];
        }
        delete[] data;
        data = new_data;
    }
    data[capacity] = elem;
    capacity++;
};

```

```

template <class T>
unsigned int Vector<T>::size() const
{
    return capacity;
}

```

```

template <class T>
void Vector<T>::erase(const int &idx)
{
    for (size_t i = idx; i < capacity - 1; ++i)
        data[i] = data[i + 1];

    capacity--;
}

```

```

template <class T>
Vector<T>::Vector()
{
    capacity = 0;
    maxsize = 0;
    data = 0;
}

```

```

template <class T>

```



```

Vector<T>::Vector(const int &n)
{
    capacity = n;
    maxsize = n;
    data = new T[capacity];
    assert(n, T());
}

template <class T>
Vector<T>::Vector(const int &n, T elem)
{
    capacity = n;
    maxsize = n;
    data = new T[capacity];
    assert(n, elem);
}

template <class T>
Vector<T>::~~Vector()
{
    delete[] data;
}

#endif

```

Address.hpp:

```

#ifndef ADDRESS_HPP
#define ADDRESS_HPP

// чтобы сравнивать строки без учета регистра
std::string to_lower(const std::string &str)
{
    std::string copy = str;
    for (auto &ch : copy)
        if(ch <= 'Z' && ch >= 'A')
            ch += 32;
    return copy;
}

class Address
{
public:
    Address() : apt(0), house(0) {}
    Address(const std::string &new_town, const std::string &new_street,
            const int &new_apt, const int &new_house)
    {
        town = new_town;
        street = new_street;
        apt = new_apt;
        house = new_house;
    }
}

```

```

bool operator==(Address const &rhs) const
{
    return to_lower(rhs.town) == to_lower(town) &&
        to_lower(rhs.street) == to_lower(street) &&
        rhs.apt == apt &&
        rhs.house == house;
}
// принадлежность городу
bool inTown(std::string const &aTown) const
{
    return std::equal(town.begin(), town.end(), aTown.begin(),
        [](char a, char b) {
            return tolower(a) == tolower(b);
        });
}
// принадлежность улице
bool inStreet(std::string const &aStreet) const
{
    return std::equal(street.begin(), street.end(), aStreet.begin(),
        [](char a, char b) {
            return tolower(a) == tolower(b);
        });
}
// является ли соседом
bool isNeighs(Address const &rhs) const
{
    if (!(to_lower(rhs.town) == to_lower(town) &&
        to_lower(rhs.street) == to_lower(street)))
        return false;
    return abs(rhs.apt - apt) <= 1;
}
// вывести всю информацию об одном адресе
void display() const
{
    std::cout << town << "\t" << street << "\t" << house << "\t" << apt;
}

private:
    std::string town, street; // название города, улицы
    int apt, house;          // номер дома, квартиры
};

#endif

```