Московский государственный технический

университет им. Н.Э. Баумана

Проверил:

Преподаватель каф. ИУ5

Подпись и дата:

Факультет «Информатика и система управления»

Кафедра ИУ5 «Система обработки информации и управления»

Выполнил:

Студент группы ИУ5-22Б

Яншин Т. И.

Подпись и дата:

Курс «Программирование на основе классов и шаблонов»

Отчёт по лабораторной работе №5

Наследование. Множества

## **Задание**

## Разработать шаблонный класс множество (MySet) на базе шаблонного класса вектор (MyVector) для выполнения операций над множествами (+, -, \*, +=, -=, \*=, ==) и функцию main() для его тестирования.

## Класс MySet и MyVector должны быть упакованы в отдельную статическую библиотеку. Пример того, как это может быть сделано, представлен в Приложении 1.

## Необходимо реализовать специализацию шаблонов для работы со скалярными типами (char\*, double, int и др.). Это необходимо сделать в виде специализации конкретных методов шаблонов, которые зависят от параметров шаблона. Иными словами, вместо реализации специализации шаблона полностью, необходимо специализировать конкретные методы шаблона, функционирование которых зависит от типов аргументов (например, деструкторы, сортировка и т.д.).

## Класс вектор должен быть динамическим массивом, размер которого может автоматически изменяться (увеличиваться или уменьшаться) в процессе выполнения программы. Добавление элементов производится в конец вектора.

## Для ускорения выполнения операций над множествами вектор, используемый классом множество, должен быть отсортирован (сортировку достаточно делать только при добавлении элемента в множество). Для поиска элементов множества следует использовать метод половинного деления.

## Методы add\_element() и delete\_element() производного класса MySet переопределяют одноименные методы базового класса MyVector, а остальные элементы класса MyVector наследуются классом MySet.

## При выполнении лабораторной работы использовать файлы с описанием классов MyVector, MySet и функцию main() главного проекта, приведенные в Приложении 1.

## **Текст программы**

Внешний CMakeLists.txt

cmake\_minimum\_required(VERSION 3.23)

set(project "lab5")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

main.cpp)

add\_subdirectory(containers)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES})

add\_executable(**${project}**

${**${project}**\_SOURCE\_LIST} )

target\_link\_libraries(**${project}** containers)

Внутренний containers /CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)

set(project "containers")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

MyVector.cpp

MySet.cpp)

set(**${project}**\_HEADERS

MyVector.h

MySet.h)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES}

${**${project}**\_HEADERS})

add\_library(**${project}**

STATIC

${**${project}**\_SOURCE\_LIST})

Main.cpp

#include "containers/MySet.h"

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

MySet<int> set;

set.add\_element(1);

set.add\_element(2);

set.add\_element(3);

set.add\_element(4);

set.add\_element(5);

std::cout << set << std::endl;

MySet<int> set2(5);

std::cout << set2 << std::endl;

MySet<int> set3 = set - set2;

std::cout << set3 << std::endl;

MySet<int> tempSet(10);

MySet<int> set4 = set3 + tempSet;

std::cout << set4 << std::endl;

MySet<int> five;

five.add\_element(4);

five.add\_element(5);

five.add\_element(6);

five.add\_element(7);

five.add\_element(8);

five.add\_element(9);

std::cout << five << std::endl;

MySet<int> out = five \* set;

std::cout << out << std::endl;

MyVector<char \*> v("Hello!");

v.add\_element("Привет!");

v.add\_element("Привет!");

v.add\_element("Привет!");

v.add\_element("Привет!");

v.add\_element("Привет!");

v.delete\_element(-1);

std::cout << "Вектор v: " << v << std::endl;

v.add\_element("Привет!");

v.add\_element("Привет!");

v.add\_element("Привет!");

std::cout << "Вектор v: " << v << std::endl;

MyVector<char \*> v1 = v;

std::cout << "Вектор v1: " << v1 << std::endl;

for (int i = 0; i < MAX\_SIZE; i++)

v1.delete\_element(0);

std::cout << "Вектор v1: " << v1 << std::endl;

std::cout << "================================" << std::endl;

MySet<char \*> s("Yes"), s1, s2;

s.add\_element("Привет!");

s.add\_element("No");

char \*str = "Hello!";

s.add\_element(str);

std::cout << "Множество s: " << s << std::endl;

s1.add\_element("Cat");

s1.add\_element("No");

s1.add\_element("Привет!");

std::cout << "Множество s1: " << s1 << std::endl;

s2 = s1 - s;

std::cout << "Множество s2=s1-s: " << s2 << std::endl;

std::cout << "Множество s1: " << s1 << std::endl;

std::cout << "Множество s: " << s << std::endl;

s2 = s - s1;

std::cout << "Множество s2=s-s1: " << s2 << std::endl;

std::cout << "Множество s1: " << s1 << std::endl;

std::cout << "Множество s: " << s << std::endl;

s2 = s1 + s;

std::cout << "Множество s2=s1+s: " << s2 << std::endl;

std::cout << "Множество s1: " << s1 << std::endl;

std::cout << "Множество s: " << s << std::endl;

s2 = s1 \* s;

std::cout << "Множество s2=s1\*s: " << s2 << std::endl;

std::cout << "Множество s1: " << s1 << std::endl;

std::cout << "Множество s: " << s << std::endl;

MySet s3 = s2;

std::cout << "Множество s3=s2: " << s3 << std::endl;

if (s3 == s2)

std::cout << "Множество s3=s2\n";

else

std::cout << "Множество s3!=s2\n";

if (s3 == s1)

std::cout << "Множество s3=s1\n";

else

std::cout << "Множество s3!=s1\n";

if (s1 == s3)

std::cout << "Множество s1=s3\n";

else

std::cout << "Множество s1!=s3\n";

return 0;

}

MyVector.h

#include <iostream>

#include <cmath>

const int MAX\_SIZE = 5;

template <typename T>

class MyVector

{

protected:

size\_t max\_size;

size\_t size;

T\* pdata;

public:

MyVector(T el = NULL , int max\_size = MAX\_SIZE);

MyVector(MyVector& vecToCopy);

~MyVector();

void add\_element(T el);

bool delete\_element(int index);

size\_t find(T el);

void resize();

void sort();

size\_t get\_size() { return size; }

size\_t get\_max\_size() { return max\_size; }

MyVector& operator=(MyVector& v);

char\* operator[](int i);

template<typename T>

friend std::ostream& operator<<(std::ostream& out, const MyVector<T>& v);

};

size\_t CalculateNewSize(int size, int max\_size)

{

if (size == max\_size || size \* 4 == max\_size)

{

return std::max(size \* 2, MAX\_SIZE);

}

return max\_size;

}

// resize

template <typename T>

void MyVector<T>::resize()

{

size\_t new\_size = CalculateNewSize(this->size, this->max\_size);

if (new\_size == this->max\_size)

{

return;

}

T\* temp = new T[new\_size];

for (size\_t i = 0; i < size; i++)

{

temp[i] = this->pdata[i];

}

delete[] this->pdata;

this->pdata = temp;

this->max\_size = new\_size;

}

template <>

void MyVector<char\*>::resize()

{

size\_t new\_size = CalculateNewSize(this->size, this->max\_size);

if (new\_size == this->max\_size)

{

return;

}

char\*\* temp = new char\* [new\_size];

for (size\_t i = 0; i < size; i++)

{

temp[i] = new char[strlen(this->pdata[i]) + 1];

strcpy(temp[i], this->pdata[i]);

delete[] this->pdata[i];

}

delete[] this->pdata;

this->pdata = temp;

this->max\_size = new\_size;

}

// add el

template<typename T>

void MyVector<T>::add\_element(T el)

{

resize();

this->pdata[this->size] = el;

this->size++;

}

template<>

void MyVector<char\*>::add\_element(char\* el)

{

resize();

this->pdata[this->size] = new char[strlen(el) + 1];

strcpy(this->pdata[this->size], el);

this->size++;

}

// Конструктор

template <typename T>

MyVector<T>::MyVector(T el, int max\_size)

{

this->size = 0;

this->max\_size = max\_size;

this->pdata = new T[max\_size];

if (el)

{

add\_element(el);

}

}

template <>

MyVector<char\*>::MyVector(char\* el, int max\_size)

{

this->size = 0;

this->max\_size = max\_size;

this->pdata = new char\* [max\_size];

if (el)

{

add\_element(el);

}

}

// Деструктор

template <typename T>

MyVector<T>::~MyVector()

{

delete[] this->pdata;

}

template <>

MyVector<char\*>::~MyVector()

{

for (size\_t i = 0; i < size; i++)

{

delete[] pdata[i];

}

delete[] pdata;

}

// К Копирования

template <typename T>

MyVector<T>::MyVector(MyVector& vecToCopy)

{

this->max\_size = vecToCopy.max\_size;

this->size = vecToCopy.size;

this->pdata = new T[this->max\_size];

for (size\_t i = 0; i < this->size; i++)

{

this->pdata[i] = vecToCopy.pdata[i];

}

}

template <>

MyVector<char\*>::MyVector(MyVector& vecToCopy)

{

this->max\_size = vecToCopy.max\_size;

this->size = vecToCopy.size;

this->pdata = new char\* [this->max\_size];

for (size\_t i = 0; i < this->size; i++)

{

this->pdata[i] = new char[strlen(vecToCopy.pdata[i]) + 1];

strcpy(this->pdata[i], vecToCopy.pdata[i]);

}

}

// удаление

template<typename T>

bool MyVector<T>::delete\_element(int index)

{

if (index >= this->size || index < 0) {

return false;

}

for (size\_t i = index; i < this->size - 1; i++) {

this->pdata[i] = this->pdata[i + 1];

}

this->size--;

resize();

return true;

}

template<>

bool MyVector<char\*>::delete\_element(int index)

{

if (index >= this->size || index < 0)

{

return false;

}

delete[] this->pdata[index];

for (size\_t i = index; i < this->size - 1; i++)

{

this->pdata[i] = this->pdata[i + 1];

}

this->size--;

resize();

return true;

}

// поиск

template<typename T>

size\_t MyVector<T>::find(T el)

{

for (size\_t i = 0; i < this->size; i++) {

if (el == pdata[i]) {

return i;

}

}

return -1;

}

template<>

size\_t MyVector<char\*>::find(char\* el)

{

for (size\_t i = 0; i < this->size; i++) {

if (el == pdata[i]) {

return i;

}

}

return -1;

}

// сортировка

template <typename T>

void MyVector<T>::sort()

{

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size - 1; j++)

{

if (pdata[j] > pdata[j + 1]) {

std::swap(pdata[j], pdata[j + 1]);

}

}

}

}

template <>

void MyVector<char\*>::sort()

{

for (size\_t i = 0; i < size; i++)

{

for (size\_t j = 0; j < size - 1; j++)

{

if (strcmp(pdata[j], pdata[j + 1])) //str

{

std::swap(pdata[j], pdata[j + 1]);

}

}

}

}

/\* Перегрузки операторов \*/

template<typename T>

MyVector<T>& MyVector<T>::operator=(MyVector& v)

{

if (this != &v)

{

delete[] this->pdata;

this->size = v.size;

this->max\_size = v.max\_size;

this->pdata = new T[this->max\_size];

for (size\_t i = 0; i < this->size; i++)

{

this->pdata[i] = v.pdata[i];

}

}

return \*this;

}

template<typename T>

char\* MyVector<T>::operator[](int i)

{

return this->pdata[i];

}

template<typename T>

std::ostream& operator<<(std::ostream& out, const MyVector<T>& v)

{

for (size\_t i = 0; i < v.size; ++i)

{

out << v.pdata[i] << " ";

}

return out;

}

/\* Перегрузки операторов \*/

template<>

MyVector<char\*>& MyVector<char\*>::operator=(MyVector& v)

{

if (this != &v)

{

for (size\_t i = 0; i < this->size; i++) // удаление левой части

{

delete[] this->pdata[i];

}

delete[] this->pdata;

this->size = v.size;

this->max\_size = v.max\_size;

this->pdata = new char\* [this->max\_size];

for (size\_t i = 0; i < this->size; i++)

{

this->pdata[i] = new char[strlen(v.pdata[i]) + 1]; // +1 \n

strcpy(this->pdata[i], v.pdata[i]);

}

}

return \*this;

}

template<>

char\* MyVector<char\*>::operator[](int i)

{

return this->pdata[i];

}

template<>

std::ostream& operator<<(std::ostream& out, const MyVector<char\*>& v)

{

for (size\_t i = 0; i < v.size; ++i)

{

out << v.pdata[i] << " ";

}

return out;

}

MyVector.cpp

#include "MyVector.h"

MySet.h

#include "MyVector.h"

template <typename T>

class MySet : public MyVector<T>

{

public:

//MySet(char\* el = NULL) : MyVector(el) {};

MySet(T el = 0) : MyVector<T>(el) {};

bool operator==(MySet& s);

MySet& operator+=(MySet& s);

MySet& operator-=(MySet& s);

MySet& operator\*=(MySet& s);

void add\_element(T el);

void delete\_element(T el);

bool is\_element(T el);

friend std::ostream& operator<< <> (std::ostream& out, MySet<T>& s);

template<typename T>

friend MySet<T> operator+(MySet<T>& s1, MySet<T>& s2);

template<typename T>

friend MySet<T> operator-(MySet<T>& s1, MySet<T>& s2);

template<typename T>

friend MySet<T> operator\* (MySet<T>& s1, MySet<T>& s2);

};

template <typename T>

std::ostream& operator<<<>(std::ostream& out, MySet<T>& s)

{

out << "{ ";

for (size\_t i = 0; i < s.size; i++) {

out << s.pdata[i] << ' ';

}

out << '}';

return out;

}

void MySet<char\*>::delete\_element(char\* el)

{

size\_t index = find(el);

if (index != -1) {

std::copy(this->pdata + index + 1, this->pdata + this->size, this->pdata + index);

this->size--;

}

}

template <typename T>

bool MySet<T>::operator==(MySet<T>& s)

{

if (this->size != s.size)

return false;

for (size\_t i = 0; i < this->size; ++i) {

if (!this->is\_element(s[i]))

return false;

}

return true;

}

template <typename T>

MySet<T>& MySet<T>::operator+=(MySet<T>& s)

{

for (size\_t i = 0; i < s.size; ++i)

this->add\_element(s[i]);

return \*this;

}

template <typename T>

MySet<T>& MySet<T>::operator-=(MySet<T>& s)

{

for (size\_t i = 0; i < s.size; ++i)

this->delete\_element(s[i]);

return \*this;

}

template <typename T>

MySet<T>& MySet<T>::operator\*=(MySet<T>& s)

{

for (size\_t i = 0; i < this->size; ++i) {

if (!s.is\_element((\*this)[i]))

this->delete\_element((\*this)[i]);

}

return \*this;

}

template <typename T>

void MySet<T>::add\_element(T el)

{

if (is\_element(el))

{

return;

}

MyVector<T>::add\_element(el);

this->sort();

}

template <typename T>

void MySet<T>::delete\_element(T el)

{

size\_t index = find(el);

if (index != -1) {

std::copy(this->pdata + index + 1, this->pdata + this->size, this->pdata + index);

this->size--;

}

}

template <typename T>

bool MySet<T>::is\_element(T el)

{

for (size\_t i = 0; i < this->size; ++i)

{

if (this->pdata[i] == el)

return true;

}

return false;

}

template <>

bool MySet<char\*>::is\_element(char\* el)

{

for (size\_t i = 0; i < this->size; ++i)

{

if (strcmp(this->pdata[i], el) == 0)

return true;

}

return false;

}

template <typename T>

MySet<T> operator+(MySet<T>& s1, MySet<T>& s2)

{

MySet set(s1);

for (size\_t i = 0; i < s2.size; i++)

{

set.add\_element(s2.pdata[i]);

}

return set;

}

template<typename T>

MySet<T> operator-(MySet<T>& s1, MySet<T>& s2)

{

MySet<T> set(s1);

for (size\_t i = 0; i < s2.size; i++)

{

set.delete\_element(s2.pdata[i]);

}

return set;

}

template <typename T>

MySet<T> operator\* (MySet<T>& s1, MySet<T>& s2)

{

MySet<T> set;

for (size\_t i1 = 0, i2 = 0; i1 < s1.size && i2 < s2.size;) {

if (s1.pdata[i1] != s2.pdata[i2]) {

if (s1.pdata[i1] < s2.pdata[i2]) {

i1++;

}

else {

i2++;

}

}

else {

set.add\_element(s1.pdata[i1]);

i1++, i2++;

}

}

return set;

}

template<>

MySet<char\*> operator\*(MySet<char\*>& s1, MySet<char\*>& s2)

{

MySet<char\*> set;

for (size\_t i1 = 0, i2 = 0; i1 < s1.size && i2 < s2.size;)

{

if (strcmp(s1.pdata[i1], s2.pdata[i2]) != 0)

{

if (strcmp(s1.pdata[i1], s2.pdata[i2]) < 0)

{

i1++;

}

else

{

i2++;

}

}

else {

set.add\_element(s1.pdata[i1]);

i1++, i2++;

}

}

return set;

}

template <typename T>

MySet<T> operator\*(MySet<T>& s2)

{

MySet set{};

for (size\_t i1 = 0, i2 = 0; i1 < this->size && i2 < s2.size;) {

if (this->pdata[i1] != s2.pdata[i2]) {

if (this->pdata[i1] < s2.pdata[i2]) {

i1++;

}

else {

i2++;

}

}

else {

set.add\_element(this->pdata[i1]);

i1++, i2++;

}

}

return set;

}

MySet.cpp

#include "MySet.h"

**Тестирование**

**Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание**