**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №5

«Наследование. Множества»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-23Б |  | преподаватель каф. ИУ5 |
| Лачина Екатерина Андреевна |  | Аладин Д.В. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г.

**1 ) Задача**

1. Разработать **шаблонный класс** множество (MySet) на базе **шаблонного класса** вектор (MyVector) для выполнения операций над множествами (+, -, \*, +=, -=, \*=, ==) и функцию main() для его тестирования.
2. Класс MySet и MyVector должны быть упакованы в отдельную статическую библиотеку. Пример того, как это может быть сделано, представлен в [Приложении 1](https://iu5edu.ru/wiki/cpp2/docs/labs/lab5/Instructions/SupplementMaterial1/).
3. 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.).
4. Перед реализацией специализаций шаблонов, настоятельно рекомендуется изучить [статью о специализации шаблона и перегрузок операторов](https://iu5edu.ru/wiki/cpp2/blog/2024-04-23-template-cout/) и [статью о линковке шаблонного класса и его специализации](https://iu5edu.ru/wiki/cpp2/blog/2024-04-17-inline-template/).
5. Класс вектор должен быть динамическим массивом, размер которого может автоматически изменяться (увеличиваться или уменьшаться) в процессе выполнения программы. Добавление элементов производится в конец вектора.
6. 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.
7. Методы add\_element() и delete\_element() производного класса MySet переопределяют одноименные методы базового класса MyVector, а остальные элементы класса MyVector наследуются классом MySet.
8. При выполнении лабораторной работы использовать файлы с описанием классов MyVector, MySet и функцию main() главного проекта, приведенные в [Приложении 1](https://iu5edu.ru/wiki/cpp2/docs/labs/lab5/Instructions/SupplementMaterial1/).

**2) Листинг**

**Папка LabaPyat**

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.23)

set(project "labaPyat")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

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

myset.cpp myvector.cpp)

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

myset.h myvector.h)

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

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

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

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

STATIC

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

**myset.cpp**

пустой

**myset.h**

#ifndef INHERITANCE\_SET\_H

#define INHERITANCE\_SET\_H

#include <iostream>

#include "myvector.h"

template <class Data>

class MySet : public MyVector<Data> {

public:

MySet(Data g = NULL) : MyVector<Data>(g) {};

bool operator==(MySet& lot);

MySet& operator+=(MySet& lot);

MySet& operator-=(MySet& lot);

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

void add\_element(Data g);

bool is\_element(Data g);

friend std::ostream& operator<<(std::ostream& out, MySet& lot) {

for (int i = 0; i < lot.size; i++) out << lot.data[i] << " ";

return out;

}

template <class T>

friend MySet operator+(MySet& l1, MySet& l2);

template <class T>

friend MySet operator-(MySet& l1, MySet& l2);

template <class T>

friend MySet operator\*(MySet& l1, MySet& l2);

MySet& operator=(MySet lot);

Data get() { return this->data[0]; }

};

template <class Data>

bool MySet<Data>::operator==(MySet& lot) {

if (this->size != lot.size) {

return false;

}

else {

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

if (this->data[i] != lot.data[i]) return false;

}

}

return true;

}

template <class Data>

bool MySet<Data>::is\_element(Data g) {

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

if (g == this->data[i]) return true;

}

return false;

}

template <class Data>

MySet<Data>& MySet<Data>::operator\*=(MySet<Data>& lot) {

for (int i = 0; i < lot.size; i++) {

if (!is\_element(lot.data[i])) add\_element(lot.data[i]);

}

return \*this;

}

template <class Data>

MySet<Data>& MySet<Data>::operator-=(MySet<Data>& lot) {

for (int i = 0; i < lot.size; i++) {

if (is\_element(lot.data[i])) delete\_element(find(lot.data[i]));

}

return \*this;

}

template <class Data>

MySet<Data>& MySet<Data>::operator+=(MySet<Data>& lot) {

int i = 0;

while (this->data[i] != NULL) {

if (!lot.is\_element(this->data[i]))

this->delete\_element(i);

else

i++;

}

return \*this;

}

template <class Data>

MySet<Data> operator+(MySet<Data>& l1, MySet<Data>& l2) {

MySet<Data> temp(l1);

temp += l2;

return temp;

}

template <class Data>

MySet<Data> operator-(MySet<Data>& l1, MySet<Data>& l2) {

MySet<Data> temp(l1);

temp -= l2;

return temp;

}

template <class Data>

void MySet<Data>::add\_element(Data g) {

if (!is\_element(g)) {

if (this->size < this->max\_size) {

this->data[this->size] = g;

this->size++;

}

else {

Data\* temp = new Data[this->size + 1];

for (int i = 0; i < this->size; i++) temp[i] = this->data[i];

temp[this->size] = g;

delete[] this->data;

this->size++;

this->max\_size++;

this->data = temp;

}

this->sort();

}

}

template <class Data>

MySet<Data> operator\*(MySet<Data>& l1, MySet<Data>& l2) {

MySet<Data> temp(l1);

temp \*= l2;

return temp;

}

template <>

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

if ((\*this).find(g) == -1) { return 0; }

return 1;

}

template <>

bool MySet<char\*>::operator==(MySet<char\*>& lot) {

if (this->size != lot.size) return false;

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

if (strcmp(this->data[i], lot.data[i]) != 0) return false;

}

return true;

}

template <>

void MySet<char\*>::add\_element(char\* g) {

if (!is\_element(g)) {

if (size < max\_size) {

data[size] = new char[strlen(g) + 1];

strcpy(data[size], g);

size++;

}

else {

char\*\* temp = new char\* [max\_size + 1];

for (int i = 0; i < size; i++) {

temp[i] = new char[strlen(data[i]) + 1];

strcpy(temp[i], data[i]);

}

temp[size] = new char[strlen(g) + 1];

strcpy(temp[size], g);

max\_size++;

size++;

delete[] data;

data = temp;

}

sort();

}

}

template <>

MySet<char\*>& MySet<char\*>::operator=(MySet<char\*> lot) {

for (int i = 0; i < size; i++) delete[] data[i];

delete[] data;

this->size = lot.size;

this->max\_size = lot.size;

data = new char\* [size];

for (int i = 0; i < size; i++) {

data[i] = new char[strlen(lot.data[i]) + 1];

strcpy(data[i], lot.data[i]);

}

return \*this;

}

template <>

MySet<char\*>& MySet<char\*>::operator-=(MySet<char\*>& lot) {

for (int i = 0; i < lot.size; i++) {

if (is\_element(lot.data[i])) delete\_element(find(lot.data[i]));

}

return \*this;

}

template <>

inline MySet<char\*>& MySet<char\*>::operator += (MySet<char\*>& lt) { //////////

for (int i{}; i < lt.size; ++i) {

if ((\*this).find((lt.data)[i]) == -1) {

(\*this).add\_element(lt.data[i]);

}

}

return \*this;

}

template <>

MySet<char\*>& MySet<char\*>::operator\*=(MySet<char\*>& lot) {

for (int i = 0; i < lot.size; i++) {

if (!is\_element(lot.data[i])) add\_element(lot.data[i]);

}

return \*this;

}

template <char\*>

MySet<char\*> operator+(MySet<char\*>& l1, MySet<char\*>& l2) {

MySet<char\*> lt;

for (int i{}; i < l1.size; ++i) {

lt.add\_element(l1[i]);

}

for (int i{}; i < l2.size; ++i) {

if (lt.find(l2[i]) == -1) {

lt.add\_element(l2[i]);

}

}

return lt;

}

template <char\*>

MySet<char\*> operator\*(MySet<char\*>& l1, MySet<char\*>& l2) {

MySet<char\*> temp(l1);

temp \*= l2;

return temp;

}

template <char\*>

MySet<char\*> operator-(MySet<char\*>& l1, MySet<char\*>& l2) {

MySet<char\*> temp(l1);

temp -= l2;

return temp;

}

#endif // INHERITANCE\_SET\_H

**myvector.cpp**

**пустой**

**myvector.h**

#ifndef INHERITANCE\_VECTOR\_H

#define INHERITANCE\_VECTOR\_H

#include <iostream>

#include <string>

const int MAX\_SIZE = 5;

template <class Data>

class MyVector {

protected:

int max\_size;

int size;

Data\* data;

void resize();

public:

MyVector(Data g = NULL, int max\_size = MAX\_SIZE);

MyVector(MyVector<Data>& vec);

~MyVector();

int get\_size() { return size; }

int get\_max\_size() { return max\_size; }

int find(Data g);

void add\_element(Data g);

void delete\_element(int index);

void sort();

MyVector<Data>& operator=(MyVector<Data>& vec);

friend std::ostream& operator<<(std::ostream& out, MyVector<Data>& vec) {

for (int i = 0; i < vector.get\_size(); i++) {

out << vector.data[i] << " ";

}

return out;

};

Data operator[](int i);

Data get() { return this->data[0]; }

};

template <class Data>

MyVector<Data>::~MyVector() {

delete[] data;

}

template <class Data>

MyVector<Data>::MyVector(Data g, int max\_size) {

size = 1;

data = new Data[max\_size];

data[0] = g;

this->max\_size = max\_size;

}

template <class Data>

MyVector<Data>::MyVector(MyVector<Data>& vec) {

if (&this != vec) {

max\_size = vec.max\_size;

size = vec.size;

data = new Data[max\_size];

for (int i = 0; i < size; i++) {

data[i] = vec.data[i];

}

}

}

template <class Data>

void MyVector<Data>::add\_element(Data g) {

if (find(g) == -1) {

data[size + 1] = g;

size++;

this->resize();

}

}

template <class Data>

int MyVector<Data>::find(Data g) {

for (int i = 0; i < size; i++) {

if (data[i] == g) return i;

}

return -1;

}

template <class Data>

void MyVector<Data>::resize() {

if (size > max\_size) {

max\_size \*= 2;

}

if (size < max\_size / 4) {

max\_size /= 2;

}

Data\* temp = new Data[max\_size];

for (int i = 0; i < size; i++) {

temp[i] = data[i];

}

delete[] data;

data = temp;

}

template <class Data>

void MyVector<Data>::delete\_element(int index) {

for (int i = index; i < size - 1; i++) {

data[i] = data[i + 1];

}

size--;

this->resize();

}

template <class Data>

MyVector<Data>& MyVector<Data>::operator=(MyVector<Data>& vec) {

if (data != nullptr) {

delete[] data;

}

data = new Data[vec.max\_size];

size = vec.size;

max\_size = vec.max\_size;

for (int i = 0; i < size; i++) {

data[i] = vec.data[i];

}

return \*this;

}

template <class Data>

void MyVector<Data>::sort() {

for (int i = 0; i < size - 1; ++i) {

for (int j = i + 1; j < size; ++j) {

if (strcmp(data[i] > data[j])) {

Data temp = data[i];

data[i] = data[j];

data[j] = temp;

}

}

}

}

template <class Data>

Data MyVector<Data>::operator[](int i) {

return data[i];

}

// ДЛЯ КЛАССА <char\*> //

template <>

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

this->max\_size = max\_size;

size = 0;

data = new char\* [max\_size];

if (g != nullptr) {

data[0] = new char[strlen(g) + 1];

strcpy(data[0], g);

size++;

}

else {

data[0] = nullptr;

}

for (int i = 1; i < max\_size; ++i) {

data[i] = nullptr;

}

}

template <>

MyVector<char\*>::MyVector(MyVector<char\*>& vec) {

data = new char\* [vec.max\_size];

size = vec.size;

max\_size = vec.max\_size;

for (int i = 0; i < size; ++i) {

data[i] = new char[strlen(vec.data[i]) + 1];

strcpy(data[i], vec.data[i]);

}

}

template <>

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

for (int i = 0; i < size; i++) {

delete[] data[i];

}

delete[] data;

}

template <>

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

if (size > max\_size) {

max\_size \*= 2;

}

if (size < max\_size / 4) {

max\_size /= 2;

}

}

template <>

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

if (size < max\_size) {

data[size] = new char[strlen(g) + 1];

strcpy(data[size], g);

size++;

}

else {

char\*\* temp = new char\* [max\_size + 1];

for (int i = 0; i < size; i++) {

temp[i] = new char[strlen(data[i]) + 1];

strcpy(temp[i], data[i]);

}

temp[size] = new char[strlen(g) + 1];

strcpy(temp[size], g);

for (int i = 0; i < size; i++) delete[] data[i];

delete[] data;

data = temp;

size++;

this->resize();

}

}

template <>

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

for (int i = index; i < size - 1; i++) {

delete[] data[i];

data[i] = new char[strlen(data[i + 1]) + 1];

strcpy(data[i], data[i + 1]);

}

delete[] data[size - 1]; /// -1

data[size - 1] = nullptr; /// -1

size--;

this->resize();

}

template <>

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

for (int i = 0; i < size - 1; i++) {

for (int j = i + 1; j < size; j++) {

if (strcmp(data[i], data[j]) > 0) {

char\* temp = new char[strlen(data[i]) + 1];

strcpy(temp, data[i]);

delete[] data[i];

data[i] = new char[strlen(data[j]) + 1];

strcpy(data[i], data[j]);

delete[] data[j];

data[j] = new char[strlen(temp) + 1];

strcpy(data[j], temp);

delete[] temp;

}

}

}

}

template <>

int MyVector<char\*>::find(char\* g) {

int index{ -1 };

for (int i = 0; i < size; ++i) {

if (g != NULL) {

if (!strcmp(data[i], g)) {

index = i; break;

}

}

}

return index;

}

template <>

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

if (data != nullptr) {

for (int i = 0; i < size; i++) {

delete[] data[i];

}

delete[] data;

}

this->size = vec.size;

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

data = new char\* [size];

for (int i = 0; i < size; i++) {

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

strcpy(this->data[i], vec.data[i]);

}

return \*this;

}

template <>

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

return data[i];

}

#endif // INHERITANCE\_VECTOR\_H

**Главные файлы**

**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(labaPyat)

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

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

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

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

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

**main.cpp**

#include "labaPyat/myset.h"

#include "labaPyat/myvector.h"

int main() {

setlocale(LC\_ALL, "Russian");

char test[20];

std::cin >> test;

MyVector<char\*> v(test);

v.add\_element(test);

char test1[20];

std::cin >> test1;

v.add\_element(test1);

std::cout << "Вектор v: " << v << '\n';

v.add\_element(test);

std::cout << "Вектор v: " << v << '\n';

MyVector<char\*> v1 = v;

std::cout << "Вектор v1: " << v1 << '\n';

std::cout << v1.get\_size() << std::endl;

std::cout << "Вектор v1: " << v1 << '\n';

MySet<char\*> s(test1), s1, s2;

s.add\_element(test);

s.add\_element(test1);

char\* str = test;

s.add\_element(str);

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

s1.add\_element(test);

s1.add\_element(test1);

s1.add\_element(test);

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";

std::string test3;

std::cin >> test3;

MySet<std::string> m(test3);

std::cout << m;

return 0;

}

1. **Тестирование**

