**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчет по лабораторной работе №5

### «Наследование. Множества»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Выполнил:** | |  | **Принял:** | |
| ФИО: | \_Цыпышев Т. А.\_\_\_\_\_ |  | ФИО: | \_Правдина А. Д.\_\_\_\_ |
| Группа: | \_ИУ5-21Б\_\_\_\_\_\_\_\_\_\_ |  | Должность: | \_Преподаватель\_\_\_\_ |
| Дата: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Дата: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва, 2022 г.

**Постановка задачи**

### Одно из основных применений наследования – использование ранее разработанных и уже проверенных классов для случаев, когда разрабатываемый класс несколько отличается от имеющегося, но есть желание использовать его в качестве базового. Дополнительные трудности на этапе освоения этой технологии возникают при использовании библиотечных классов из-за того, что библиотечные классы разработаны как шаблоны классов и их использование имеет свои особенности.

Класс множество (set) является одним из типов ассоциативных контейнеров. В ассоциативных контейнерах доступ к данным выполняется не по индексу элемента (как в массиве), а по его значению (ключу). Множества хранят элементы, значения ключей которых должны быть уникальными. Наиболее часто используемой операцией при работе с множествами является поиск элемента. Поэтому для эффективной работы класс set должен обеспечивать быстрый доступ к данным по ключу. Для реализации этого требования множества, как правило, реализуются на основе сбалансированного дерева или отсортированного массива.

### Задание.

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

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*** перегружают одноименные методы базового класса ***MyVector***, а остальные элементы класса ***MyVector*** наследуются классом ***MySet*** .

#### Описание программы:

Класс с именем **MySet** включает следующие элементы:

# Члены - данные

Все данные наследуются из класса ***MyVector*** (элементы множества хранятся в векторе)***.***

**Конструкторы и деструкторы**

Так как своих данных в ***MySet*** нет, то можно не перегружать имеющиеся (по умолчанию) конструктор копирования, оператор присваивания и деструктор.  
При необходимости будут вызываться соответствующие элементы базового класса.

**Методы доступа**

**IsElement**, который даёт **true**, если строка-параметр есть в множестве, иначе даёт **false**. Для поиска элементов множества следует использовать метод половинного деления. Для его реализации разработать метод **q\_find**, который имеет тип доступа **private**.

**Методы изменения**

**AddElement** - добавляет строку в множество, если её там ещё нет. Для ускорения поиска элементов создаваемое множество должно быть отсортировано по возрастанию значения ключа. Для сортировки при добавлении элементов использовать метод **sort()** базового класса.

**DeleteElement** для удаления строки из множества, если она там есть

**Операторы**

**операторы присваивания** *-=, +=, \*=,* где - означает разность, + **-** объединение и **\*** - пересечение множеств. (См. примеры ниже.)

**Функции – не члены класса (друзья класса)**

Перегруженная операция потокового вывода;

**Операторы +** (объединение), **-** (разность), **\*** (пересечение) и **==** (сравнение: истина, если элементы двух множеств совпадают).

Примеры операций над множествами (приведены для целых чисел):

{1, 4, 5, 6} + {1, 2, 3, 4} => {1, 2, 3, 4, 5, 6}

{1, 4, 5, 6} \* {1, 2, 3, 4} => {1, 4}

{1, 4, 5, 6} - {1, 2, 3, 4} => {5, 6}

Базовый класс **MyVector**, является динамическим ***массивом строк.*** Размер вектора ***maxsize*** должен меняться в процессе выполнения программы следующим образом:

если при добавлении элемента число элементов вектора ***size*** превысит размер вектора, ***maxsize*** увеличивается примерно в 1,5 раза (был 8, станет 12, если ***size***>=8)

если при удалении элемента число элементов вектора ***size*** станет меньше ***maxsize/2,*** ***maxsize*** уменьшается примерно в 1,5 раза, но должен быть не меньше значения по умолчанию (был 12, станет 8, если ***size*** <6). Новый элемент добавляется в конец вектора.

**Члены - данные (protected)** **:**

**maxsize –**размер вектора;

**size** – количество элементов в векторе;

**pdata** – указатель, содержащий адрес динамического массива элементов (строк).

**Класс MyVector должен реализовывать следующие функции:**

**add\_element** – вставка элемента в конец вектора;

**delete\_element** – удаление элемента из произвольного места;

**find(el) –** возвращает индекс элемента или –1, если элемент не найден;

**resize** – изменение размера вектора **maxsize** при его переполнении или освобождении места **(private);**

**Конструкторы и деструктор**

Конструктор с одним параметром (символьная строка) для создания множества размером 1, который имеет значения по умолчанию и поэтому может использоваться для создания пустого множества;

Конструктор копирования;

Деструктор.

**Операторы**

**[ ]** - для возврата элемента вектора (доступ по индексу);

**=** - оператор присваивания.

При выполнении лабораторной работы использовать файлы с описанием классов ***MyVector, MySet*** и тестирующей программой ***lab4***, приведенные в Приложении 1.

Работа состоит из четырех частей: ЛР4.1, ЛР4.2, ЛР4.3, ЛР4.4. Общее время выполнения работы – 16 часов.

Содержание частей ЛР:

ЛР4.1 – Разработка и тестирование класса ***MyVector*** для хранения символьных строк.

ЛР4.2 – Разработка и тестирование класса ***MySet*** для выполнения операций над множествами символьных строк.

ЛР4.3 - Разработка и тестирование шаблонного класса ***MyVector***.

ЛР4.4 – Разработка и тестирование шаблонного класса ***MySet.***

Приложение 1.

//файл MyVector.h - описание класса MyVector

#ifndef MYVECTOR\_H

#define MYVECTOR\_H

#include <iostream>

using namespace std;

const int MAX\_SIZE = 5;

class MyVector

{

public:

MyVector(char \*el = NULL, int maxsz = MAX\_SIZE);

MyVector(MyVector& v);

~MyVector();

void add\_element(char\* el);

bool delete\_element(int i);

char\* operator[](int i);

void sort();

int Size(){return size;}

int Maxsize(){return maxsize;}

int find(char\* el);

MyVector& operator=(MyVector& v);

friend ostream& operator<<(ostream& out, MyVector& v);

protected:

int maxsize;

int size;

char \*\* pdata;

private:

void resize();

};

#endif

//файл MySet.h - описание класса MySet

#ifndef MYSET\_H

#define MYSET\_H

#include <iostream>

using namespace std;

class MySet:public MyVector

{

public:

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

friend ostream& operator<<(ostream& out, MySet& s);

friend MySet operator+(MySet& s1,MySet& s2);

friend MySet operator-(MySet& s1,MySet& s2);

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

bool operator==(MySet& s);

MySet& operator+=(MySet& s);

MySet& operator-=(MySet& s);

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

void add\_element(char\* el);

void delete\_element(char\* el);

bool is\_element(char\* el);

};

#endif

//файл lab4 для тестирования классов MyVector и MySet

#include <iostream>

#include "MyVector.h"

#include "MySet.h"

#include "MyVector.cpp"

#include "MySet.cpp"

using namespace std;

int main ()

{

setlocale(0,"russian");

MyVector v("Hello!");

v.add\_element("Привет!");

v.add\_element("Привет!");

v.add\_element("Привет!");

v.add\_element("Привет!");

v.add\_element("Привет!");

cout<<"Вектор v: "<<v<<endl;

v.add\_element("Привет!");

v.add\_element("Привет!");

v.add\_element("Привет!");

cout<<"Вектор v: "<<v<<endl;

MyVector v1=v;

cout<<"Вектор v1: "<<v1<<endl;

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

v1.delete\_element(0);

cout<<"Вектор v1: "<<v1<<endl;

MySet s("Yes"),s1,s2;

s.add\_element("Привет!");

s.add\_element("No");

char\* str="Hello!";

s.add\_element(str);

cout<<"Множество s: "<<s<<endl;

s1.add\_element("Cat");

s1.add\_element("No");

s1.add\_element("Привет!");

cout<<"Множество s1: "<<s1<<endl;

s2=s1-s;

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

cout<<"Множество s1: "<<s1<<endl;

cout<<"Множество s: "<<s<<endl;

s2=s-s1;

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

cout<<"Множество s1: "<<s1<<endl;

cout<<"Множество s: "<<s<<endl;

s2=s1+s;

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

cout<<"Множество s1: "<<s1<<endl;

cout<<"Множество s: "<<s<<endl;

s2=s1\*s;

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

cout<<"Множество s1: "<<s1<<endl;

cout<<"Множество s: "<<s<<endl;

MySet s3=s2;

cout<<"Множество s3=s2: "<<s3<<endl;

if(s3==s2)

cout<<"Множество s3=s2\n";

else

cout<<"Множество s3!=s2\n";

if(s3==s1)

cout<<"Множество s3=s1\n";

else

cout<<"Множество s3!=s1\n";

if(s1==s3)

cout<<"Множество s1=s3\n";

else

cout<<"Множество s1!=s3\n";

return 0;

}

**Текст программы**

**main.cpp**

*/// Copyright 2023 ttsypyshev <ttsypyshev01@gmail.com>*#include "MyVector.h"  
#include "MySet.h"  
#include <iostream>  
  
using namespace std;  
  
int main() {  
 MyVector<char \*> v("Hello!");  
 v.add\_element("Привет!");  
 v.add\_element("Привет!");  
 v.add\_element("Привет!");  
 v.add\_element("Привет!");  
 v.add\_element("Привет!");  
 cout << "Вектор v: " << v << endl;  
 v.add\_element("Привет!");  
 v.add\_element("Привет!");  
 v.add\_element("Привет!");  
 cout << "Вектор v: " << v << endl;  
 MyVector<char \*> v1 = v;  
 cout << "Вектор v1: " << v1 << endl;  
 for (int i = 0; i < MAX\_SIZE; i++)  
 v1.delete\_element(0);  
 cout << "Вектор v1: " << v1 << endl;  
 MySet<char \*> s("Yes"), s1, s2;  
 s.add\_element("Привет!");  
 s.add\_element("No");  
 char \*str = "Hello!";  
 s.add\_element(str);  
 cout << "Множество s: " << s << endl;  
 s1.add\_element("Cat");  
 s1.add\_element("No");  
 s1.add\_element("Привет!");  
 cout << "Множество s1: " << s1 << endl;  
 s2 = s1 - s;  
 cout << "Множество s2=s1-s: " << s2 << endl;  
 cout << "Множество s1: " << s1 << endl;  
 cout << "Множество s: " << s << endl;  
 s2 = s - s1;  
 cout << "Множество s2=s-s1: " << s2 << endl;  
 cout << "Множество s1: " << s1 << endl;  
 cout << "Множество s: " << s << endl;  
 s2 = s1 + s;  
 cout << "Множество s2=s1+s: " << s2 << endl;  
 cout << "Множество s1: " << s1 << endl;  
 cout << "Множество s: " << s << endl;  
 s2 = s1 \* s;  
 cout << "Множество s2=s1\*s: " << s2 << endl;  
 cout << "Множество s1: " << s1 << endl;  
 cout << "Множество s: " << s << endl;  
 MySet<char \*> s3 = s2;  
 cout << "Множество s3=s2: " << s3 << endl;  
 if (s3 == s2)  
 cout << "Множество s3=s2\n";  
 else  
 cout << "Множество s3!=s2\n";  
 if (s3 == s1)  
 cout << "Множество s3=s1\n";  
 else  
 cout << "Множество s3!=s1\n";  
 if (s1 == s3)  
 cout << "Множество s1=s3\n";  
 else  
 cout << "Множество s1!=s3\n";  
 return 0;  
}

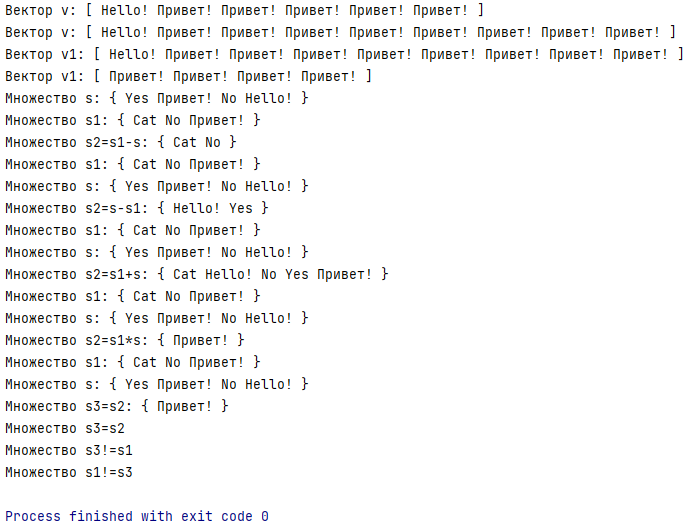
**MyVector.h**

*/// Copyright 2023 ttsypyshev <ttsypyshev01@gmail.com>*#ifndef **INF\_LAB\_5\_MYVECTOR\_H**#define **INF\_LAB\_5\_MYVECTOR\_H**#include <iostream>  
#include <cmath>  
#include <cstring>  
#include <algorithm>  
  
const int MAX\_SIZE = 5;  
  
template<class TYPE>  
class MyVector {  
protected:  
 int maxsize;  
 int size;  
 TYPE \*pdata;  
private:  
 void resize();  
  
public:  
 MyVector();  
 MyVector(TYPE element);  
 MyVector(MyVector<TYPE> &other);  
 ~MyVector();  
  
 void add\_element(TYPE element);  
 bool delete\_element(int index);  
 int find\_element(TYPE element);  
 void sort();  
  
 friend std::ostream &operator<<(std::ostream &out, MyVector<TYPE> &other) {  
 out << "[ ";  
 for (size\_t i = 0; i < other.size; ++i) {  
 out << other[i] << " ";  
 }  
 out << "]";  
 return out;  
 }  
  
 const TYPE &operator[](size\_t index) const;  
 MyVector<TYPE> &operator=(const MyVector<TYPE> &other);  
};  
  
template<class TYPE>  
void MyVector<TYPE>::resize() {  
 if (size >= maxsize) { maxsize = static\_cast<int>(maxsize \* 1.5); }  
 else if (size < maxsize / 2) { maxsize = std::max(static\_cast<int>(maxsize \* 0.5), MAX\_SIZE); }  
 else { return; }  
  
 TYPE \*new\_pdata = new TYPE[maxsize];  
 for (size\_t i = 0; i < size; ++i) {  
 new\_pdata[i] = pdata[i];  
 }  
 delete[] pdata;  
 pdata = new\_pdata;  
}  
  
template<class TYPE>  
MyVector<TYPE>::MyVector() {  
 maxsize = MAX\_SIZE;  
 size = 0;  
 pdata = new TYPE[MAX\_SIZE];  
}  
  
template<class TYPE>  
MyVector<TYPE>::MyVector(TYPE element) {  
 maxsize = MAX\_SIZE;  
 size = 0;  
 pdata = new TYPE[MAX\_SIZE];  
 pdata[size++] = element;  
}  
  
template<>  
MyVector<char \*>::MyVector(char \*element) {  
 maxsize = MAX\_SIZE;  
 size = 0;  
 pdata = new char \*[MAX\_SIZE];  
 pdata[0] = new char[strlen(element) + 1];  
 strcpy(pdata[size++], element);  
}  
  
template<class TYPE>  
MyVector<TYPE>::MyVector(MyVector<TYPE> &other) {  
 maxsize = other.maxsize;  
 size = other.size;  
 pdata = new TYPE[maxsize];  
 for (size\_t i = 0; i < size; ++i) {  
 pdata[i] = other[i];  
 }  
}  
  
template<>  
MyVector<char \*>::MyVector(MyVector<char \*> &other) {  
 maxsize = other.maxsize;  
 size = other.size;  
 pdata = new char \*[maxsize];  
 for (int i = 0; i < size; ++i) {  
 pdata[i] = new char[strlen(other[i]) + 1];  
 strcpy(pdata[i], other[i]);  
 }  
}  
  
template<class TYPE>  
MyVector<TYPE>::~MyVector() {  
 delete[] pdata;  
}  
  
template<>  
MyVector<char \*>::~MyVector() {  
 for (size\_t i = 0; i < size; ++i) {  
 delete[] pdata[i];  
 }  
 delete[] pdata;  
}  
  
template<class TYPE>  
void MyVector<TYPE>::add\_element(TYPE element) {  
 pdata[size++] = element;  
 resize();  
}  
  
template<>  
void MyVector<char \*>::add\_element(char \*element) {  
 pdata[size] = new char[strlen(element) + 1];  
 strcpy(pdata[size++], element);  
 resize();  
}  
  
template<class TYPE>  
bool MyVector<TYPE>::delete\_element(int index) {  
 if (index >= 0 && index <= size--) {  
 for (size\_t i = index; i < size; ++i) {  
 pdata[i] = pdata[i + 1];  
 }  
 resize();  
 return true;  
 }  
 return false;  
}  
  
template<class TYPE>  
int MyVector<TYPE>::find\_element(TYPE element) {  
 for (int i = 0; i < size; ++i) {  
 if (pdata[i] == element) return i;  
 }  
 return -1;  
}  
  
template<>  
int MyVector<char \*>::find\_element(char \*element) {  
 for (size\_t i = 0; i < size; ++i) {  
 if (strlen(pdata[i]) == strlen(element)) {  
 if (strcmp(pdata[i], element) == 0) return i;  
 }  
 }  
 return -1;  
}  
  
template<class TYPE>  
void MyVector<TYPE>::sort() {  
 std::sort(pdata, pdata + size);  
}  
  
template<>  
void MyVector<char \*>::sort() {  
 std::sort(pdata, pdata + size, [](const char \*a, const char \*b) {  
 return std::strcmp(a, b) < 0;  
 });  
}  
  
template<class TYPE>  
const TYPE &MyVector<TYPE>::operator[](size\_t index) const {  
 return pdata[index];  
}  
  
template<class TYPE>  
MyVector<TYPE> &MyVector<TYPE>::operator=(const MyVector<TYPE> &other) {  
 if (this == &other) return \*this;  
  
 maxsize = other.maxsize;  
 size = other.size;  
 delete[] pdata;  
 pdata = new TYPE[maxsize];  
 for (size\_t i = 0; i < size; ++i) {  
 pdata[i] = other[i];  
 }  
  
 return \*this;  
}  
  
template<>  
MyVector<char \*> &MyVector<char \*>::operator=(const MyVector<char \*> &other) {  
 if (this == &other) return \*this;  
  
 maxsize = other.maxsize;  
 size = other.size;  
 delete[] pdata;  
 pdata = new char \*[maxsize];  
 for (int i = 0; i < size; ++i) {  
 pdata[i] = new char[strlen(other[i]) + 1];  
 strcpy(pdata[i], other[i]);  
 }  
  
 return \*this;  
}  
  
#endif *//INF\_LAB\_5\_MYVECTOR\_H*

**MySet.h**

*/// Copyright 2023 ttsypyshev <ttsypyshev01@gmail.com>*#ifndef **INF\_LAB\_5\_MYSET\_H**#define **INF\_LAB\_5\_MYSET\_H**#include <iostream>  
#include "MyVector.h"  
  
template<class TYPE>  
class MySet : public MyVector<TYPE> {  
private:  
 int q\_find(TYPE element);  
public:  
 MySet() : MyVector<TYPE>() {}  
 MySet(TYPE element) : MyVector<TYPE>(element) {}  
 MySet(MySet<TYPE> &other) : MyVector<TYPE>(other) {}  
  
 bool IsElement(TYPE element);  
 void AddElement(TYPE element);  
 bool DeleteElement(TYPE element);  
  
 MySet<TYPE> operator+(MySet<TYPE> &other);  
 MySet<TYPE> operator-(MySet<TYPE> &other);  
 MySet<TYPE> operator\*(MySet<TYPE> &other);  
*// MySet<TYPE> &operator+=(MySet<TYPE> &other);  
// MySet<TYPE> &operator-=(MySet<TYPE> &other);  
// MySet<TYPE> &operator\*=(MySet<TYPE> &other);* friend std::ostream &operator<<(std::ostream &out, MySet<TYPE> &other) {  
 out << "{ ";  
 for (size\_t i = 0; i < other.size; ++i) {  
 out << other[i] << " ";  
 }  
 out << "}";  
 return out;  
 }  
  
 friend bool operator==(MySet<TYPE> &set1, MySet<TYPE> &set2) {  
 if (set1.size != set2.size)  
 return false;  
  
 for (int i = 0; i < set1.size; ++i) {  
 if (!set2.IsElement(set1[i]))  
 return false;  
 }  
 return true;  
 }  
};  
  
template<class TYPE>  
int MySet<TYPE>::q\_find(TYPE value) {  
 int left = 0;  
 int right = this->size - 1;  
  
 while (left <= right) {  
 int middle = (left + right) / 2;  
 if (value < (\*this)[middle]) {  
 right = middle - 1;  
 } else if (value > (\*this)[middle]) {  
 left = middle + 1;  
 } else {  
 return middle;  
 }  
 }  
 return -1;  
}  
  
template<>  
int MySet<char \*>::q\_find(char \*value) {  
 int left = 0;  
 int right = this->size - 1;  
  
 while (left <= right) {  
 int middle = (left + right) / 2;  
 if (strcmp(value, (\*this)[middle]) < 0) {  
 right = middle - 1;  
 } else if (strcmp(value, (\*this)[middle]) > 0) {  
 left = middle + 1;  
 } else {  
 return middle;  
 }  
 }  
 return -1;  
}  
  
template<class TYPE>  
bool MySet<TYPE>::IsElement(TYPE element) {  
 return q\_find(element) != -1;  
}  
  
template<class TYPE>  
void MySet<TYPE>::AddElement(TYPE element) {  
 if (!IsElement(element)) {  
 this->add\_element(element);  
 this->sort();  
 }  
}  
  
template<class TYPE>  
bool MySet<TYPE>::DeleteElement(TYPE element) {  
 int index = q\_find(element);  
 return this->delete\_element(index);  
}  
  
template<class TYPE>  
MySet<TYPE> MySet<TYPE>::operator-(MySet<TYPE> &other) {  
 MySet<TYPE> result;  
 for (int i = 0; i < this->size; ++i) {  
 if (other.q\_find(this->pdata[i]) == -1)  
 result.AddElement(this->pdata[i]);  
 }  
 return result;  
}  
  
template<class TYPE>  
MySet<TYPE> MySet<TYPE>::operator+(MySet<TYPE> &other) {  
 MySet<TYPE> result(\*this);  
 for (int i = 0; i < other.size; ++i) {  
 result.AddElement(other[i]);  
 }  
 return result;  
}  
  
template<class TYPE>  
MySet<TYPE> MySet<TYPE>::operator\*(MySet<TYPE> &other) {  
 MySet<TYPE> result;  
 for (int i = 0; i < this->size; ++i) {  
 if (other.IsElement(this->pdata[i]))  
 result.AddElement(this->pdata[i]);  
 }  
 return result;  
}  
  
#endif *//INF\_LAB\_5\_MYSET\_H*

**Тестирование функций**

****