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 №7

Виртуальные функции

## **Задание**

Задание

Иерархия классов

Для определения иерархии классов связать отношением наследования классы, приведенные в приложении (для заданного варианта). Из перечисленных классов выбрать один, который будет стоять во главе иерархии. Это абстрактный класс.

Определить в классах все необходимые конструкторы и деструкторы. Не забыть в абстрактном классе определить виртуальный деструктор.

В конструкторах и деструкторах для абстрактного класса и производных классов предусмотреть вывод отладочных сообщений об их вызове в консоль. Например:

std::cout << "A()" << std::endl; // при вызове конструктора

std::cout << "~A()" << std::endl; // при вызове деструктора

Компонентные данные класса специфицировать как protected.

В абстрактном классе объявить виртуальный метод show, который отвечает за отображение данных класса. В каждом производном классе реализовать данный метод с учетом специфики каждого класса.

Определение классов, их реализацию, демонстрационную программу поместить в отдельные файлы. Данные файлы должны быть упакованы в отдельную статическую библиотеку.

Контейнер классов

В функции main() создать экземпляр самостоятельно разработанного вектора из предыдущей работы для реализации контейнера из заданных классов (далее – контейнер). В качестве типа аргумента шаблона использовать указатель на абстрактный класс для демонстрации полиморфизма.

ОСТОРОЖНО

В работе запрещено использование контейнеров STL (vector, list и др.).

Перед использованием самостоятельно разработанного вектора рекомендуется изучить статью о линковке шаблонного класса и его специализации.

Реализовать функции, которые в качестве аргумента принимают ссылку на контейнер:

Функция print для вывода всех связанных с контейнером экземпляров производных классов с указанием индекса в контейнере.

Функция remove для удаления конкретного указателя по заданному индексу из контейнера и для удаления динамического объекта по этому указателю.

Функция clear для очистки контейнера и удаления всех динамически созданных объектов по указателям из контейнера.

К СВЕДЕНИЮ

При отладке программы по отладочным сообщениям из деструкторов, убедитесь в корректной работе виртуальных деструкторов.

Добавление в контейнер динамически созданных экземпляров заданных классов.

Работу функции print.

Работу функции remove.

Работу функции clear.

## **Текст программы**

Внешний CMakeLists.txt

cmake\_minimum\_required(VERSION 3.23)

set(project "lab7")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 20)

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

main.cpp

Ship.h

Steamboat.h

Sailboat.h

Сorvette.h

)

add\_subdirectory("MyVector")

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

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

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

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

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

Внутренний MyVector/CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)

set(project "MyVector")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 20)

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

MyVector.cpp)

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

MyVector.h)

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

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

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

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

STATIC

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

Main.cpp

#include <iostream>

#include "MyVector/MyVector.h"

#include "Ship.h"

#include "Steamboat.h"

#include "Sailboat.h"

#include "Сorvette.h"

static void print(MyVector<Ship\*>& ships) {

std::cout << "\n";

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

std::cout << "Индекс судна: " << i << std::endl;

ships[i]->show();

std::cout << std::endl;

}

}

static void remove(MyVector<Ship\*>& ships, int index) {

ships.delete\_element(index);

}

static void clear(MyVector<Ship\*>& ships) {

while (ships.get\_size() > 0) {

ships.delete\_element(0);

}

}

static void add\_ship\_interactive(MyVector<Ship\*>& ships) {

int choice;

std::string name;

int displacement, some\_other\_param;

std::cout << "Выберите тип судна (1 - Steamboat, 2 - Sailboat, 3 - Сorvette): ";

std::cin >> choice;

std::cout << "Введите имя судна: ";

std::cin >> name;

std::cout << "Введите водоизмещение: ";

std::cin >> displacement;

switch (choice) {

case 1:

std::cout << "Введите количество пассажиров: ";

break;

case 2:

std::cout << "Введите площадь парусов: ";

break;

case 3:

std::cout << "Введите максимальную скорость: ";

break;

default:

std::cout << "Неверный выбор!" << std::endl;

exit(1);

}

std::cin >> some\_other\_param;

switch (choice) {

case 1:

ships.add\_element(new Steamboat((char\*)name.c\_str(), displacement, some\_other\_param));

break;

case 2:

ships.add\_element(new Sailboat((char\*)name.c\_str(), displacement, some\_other\_param));

break;

case 3:

{

ships.add\_element(new Сorvette((char\*)name.c\_str(), displacement, some\_other\_param));

break;

}

default:

std::cout << "Неверный выбор!" << std::endl;

exit(1);

}

}

static void interactive\_mode() {

MyVector<Ship\*> ships;

int choice;

int index;

while (true) {

std::cout << "Меню:\n";

std::cout << "1. Добавить судно\n";

std::cout << "2. Печать судов\n";

std::cout << "3. Удалить судно\n";

std::cout << "4. Очистить все суда\n";

std::cout << "5. Выйти\n";

std::cout << "Выберите действие: ";

std::cin >> choice;

switch (choice) {

case 1:

add\_ship\_interactive(ships);

break;

case 2:

print(ships);

break;

case 3:

std::cout << "Введите индекс судна для удаления: ";

std::cin >> index;

if (index >= 0 && index < ships.get\_size()) {

remove(ships, index);

}

else {

std::cout << "Неверный индекс!" << std::endl;

}

break;

case 4:

clear(ships);

break;

case 5:

clear(ships);

return;

default:

std::cout << "Неверный выбор!" << std::endl;

}

}

}

static void demo\_mode() {

MyVector<Ship\*> ships;

ships.add\_element(new Steamboat((char\*)"Aurora", 1500, 30));

ships.add\_element(new Steamboat((char\*)"Emperor", 1800, 35));

ships.add\_element(new Sailboat((char\*)"Seabreeze", 3500, 50));

ships.add\_element(new Sailboat((char\*)"Windrider", 4200, 55));

ships.add\_element(new Сorvette((char\*)"Vigilant", 2500, 40));

ships.add\_element(new Сorvette((char\*)"Defender", 2700, 45));

std::cout << "Демонстрация работы функции print:\n";

print(ships);

std::cout << "(удаление судна с индексом 2):\n";

remove(ships, 2);

print(ships);

std::cout << "(очистка всех судов):\n";

clear(ships);

}

int main() {

setlocale(LC\_ALL, "Russian");

int mode;

std::cout << "Выберите режим работы (1 - интерактивный, 2 - демонстрационный): ";

std::cin >> mode;

if (mode == 1) {

interactive\_mode();

}

else if (mode == 2) {

demo\_mode();

}

else {

std::cout << "Неверный выбор!" << std::endl;

}

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);

void clear();

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; }

T\* begin() { return pdata; }

T\* end() { return pdata + size; }

bool empty() const { return size == 0; }

MyVector& operator=(MyVector& v);

//char\* operator[](int i);

T operator[](int index);

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;

}

template <typename T>

T MyVector<T>::operator[](int index)

{

return pdata[index];

}

template <>

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

{

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

{

delete[] pdata[i];

pdata[i] = nullptr;

}

size = 0;

delete[] pdata;

pdata = nullptr;

}

template <typename T>

void MyVector<T>::clear()

{

size = 0;

delete[] pdata;

pdata = nullptr;

}

// 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];

}

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;

}

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;

}

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) {

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>

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)

{

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 for null terminator

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"

Ship.h

#pragma once

#include <iostream>

class Ship {

protected:

char\* name;

int waterDisplacement;

public:

explicit Ship(char\* name, int waterDisplacement) {

this->name = new char[strlen(name) + 1];

strcpy(this->name, name);

this->waterDisplacement = waterDisplacement;

std::cout << "Ship()" << std::endl;

}

virtual ~Ship() {

std::cout << "~Ship()" << std::endl;

delete[] name;

};

virtual void show() = 0;

};

Steamboat.h

#pragma once

#include "Ship.h"

class Steamboat : public Ship {

private:

int passangers;

public:

Steamboat(char\* name, int waterDisplacement, int passangers) : Ship(name, waterDisplacement) {

this->passangers = passangers;

std::cout << "Steamboat()" << std::endl;

}

~Steamboat() override {

std::cout << "~Steamboat()" << std::endl;

}

void show() override {

std::cout << "Пароход" << std::endl;

std::cout << "Название: " << name << std::endl;

std::cout << "Водоизмещение:" << waterDisplacement << std::endl;

std::cout << "Кол-во пассажирских мест: " << passangers << std::endl;

}

};

Sailboat.h

#pragma once

#include "Ship.h"

class Sailboat : public Ship {

private:

int sailsArea;

public:

Sailboat(char\* name, int waterDisplacement, int sailsArea) : Ship(name, waterDisplacement) {

this->sailsArea = sailsArea;

std::cout << "Sailboat()" << std::endl;

}

~Sailboat() override {

std::cout << "~Sailboat()" << std::endl;

}

void show() override {

std::cout << "Парусник" << std::endl;

std::cout << "Название: " << name << std::endl;

std::cout << "Водризмещение: " << waterDisplacement << std::endl;

std::cout << "Площадь парусов: " << sailsArea << std::endl;

}

};

Сorvette.h

#pragma once

#include "Ship.h"

class Сorvette : public Ship {

private:

int maxSpeed;

public:

Сorvette(char\* name, int waterDisplacement, int maxSpeed) : Ship(name, waterDisplacement) {

this->maxSpeed = maxSpeed;

std::cout << "Сorvette()" << std::endl;

}

~Сorvette() override {

std::cout << "~Сorvette()" << std::endl;

}

void show() override {

std::cout << "Корвет" << std::endl;

std::cout << "Название: " << name << std::endl;

std::cout << "Водоизмещение: " << waterDisplacement << std::endl;

std::cout << "Максимальная скорость: " << maxSpeed << std::endl;

}

};

**Тестирование**

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |
| --- |
|  |
|  |
|  |