Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет: «Информационных технологий и прикладной математики»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 6**

**Тема: Основы работы с коллекциями: аллокаторы**

|  |  |
| --- | --- |
| Cтудент: | Королев И.М. |
| Группа: | 8О-208Б |
| Преподаватель: | Чернышов Л.Н. |
| Дата: |  |
| Оценка: |  |

Москва 2020

Оглавление

[1. Постановка задачи 3](#_Toc58234416)

[2. Описание программы 4](#_Toc58234417)

[3. Тесты и результаты их выполнения 7](#_Toc58234424)

[5. Листинг программы 10](#_Toc58234425)

[6. Выводы 19](#_Toc58234426)

[Список используемых источников 20](#_Toc58234427)

# 1. Постановка задачи

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных, задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения, т.е. равносторонними (кроме трапеции и прямоугольника). Для хранения координат фигур необходимо использовать шаблон std::pair.

Создать шаблон динамической коллекции, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr). Опционально использование std::unique\_ptr;
2. В качестве параметра шаблона коллекция должна принимать тип данных - фигуры;
3. Коллекция должна содержать метод доступа:

* Стек – pop, push, top;
* Очередь – pop, push, top;
* Список, динамический массив – доступ к элементу по оператору [];

1. Реализовать аллокатор, который выделяет фиксированный размер памяти (количество блоков памяти – является параметром шаблона аллокатора). Внутри аллокатор должен хранить указатель на используемый блок памяти и динамическую коллекцию указателей на свободые блоки. Динамическая коллекция должна соответствовать варианту задания (динамический массив, список, стек, очередь);
2. Коллекция должна использовать аллокатор для выделения и освобождения памяти для своих элементов.
3. Аллокатор должен быть совместим с контейнерами std::map и std::list (опционально - vector).
4. Реализовать программу, которая:

* Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию, использующую аллокатор;
* Позволяет удалять элемент из коллекции по его номеру;
* Выводит на экран выведенные фигуры с помощью std::for\_each;

Таблица 1 – вариант задания

|  |  |  |  |
| --- | --- | --- | --- |
| **Вариант** | **Фигура** | **Контейнер** | **Аллокатор** |
| 1 | Треугольник | Стек | Динамический массив |

# 2. Описание программы

Программа выполняет чтение данных из консоли или файла (с помощью cat | ./исполняемый файл) и выполняет заданные действия. При запуске программы появляется меню, в котором будут предложены следующие варианты действий:

1. Добавить в стек новую фигуру.
2. Добавить в стек фигуру на заданную позицию.
3. Удалить фигуру из стека.
4. Удалить из стека фигуру, находящуюся на заданной позиции.
5. Вывести координаты фигур, находящихся в стеке.
6. Завершить программу.

Коллекция (стек) была реализована с помощью умных указателей std::shared\_ptr. Умный указатель std::shared\_ptr предназначен для случаев, когда несколько умных указателей совместно владеют одним динамическим ресурсом. Он отслеживает количество владельцев у каждого полученного ресурса. До тех пор, пока хотя бы один std::shared\_ptr владеет ресурсом, этот ресурс не будет уничтожен, даже если были удалены остальные std::shared\_ptr, которые тоже владели этим ресурсом.

При удалении несуществующего элемента или добавлении элемента на позицию в очереди, которая противоречит условиям, были сгенерированы исключения с помощью try и catch. Блок try обнаруживает любые исключения, которые были выброшены в нём, и направляет в блок catch для обработки. Блок try должен иметь хотя бы один блок catch, который находится сразу же за ним. После выполнения блока catch программа возобновляется. Добавление и удаление фигуры из списка по индексу выполняется с помощью итераторов.

Коллекция стек использует аллокатор для выделения и освобождения памяти для своих элементов. Аллокатор содержит в себе указатель на используемый блок памяти и динамический массив, в котором содержатся указатели на свободные блоки. При выделении памяти, указатели на блоки, которые были выделены, удаляются из динамического массива.

Вывод всех фигур, находящихся в стеке, производится с помощью std::for\_each().

# Структура Triangle

*std::pair<T,T> Center*  – координаты геометрического центра треугольника.

*T Side* – сторона треугольника.

*Triangle& operator=(const Triangle &other)* – оператор присваивания значений одного треугольника другому.

std::ostream& operator << (std::ostream& cout, const Triangle<T> &object) – оператор вывода координат треугольника.

# Класс allocator

*T\* Buffer* – указатель на используемый блок памяти.

*std::vector<T\*> Vector* – динамическая коллекция указателей на свободные блоки.

*T \*allocate(const std::size\_t &size)* – функция выделения памяти для заданного размера.

*struct rebind* – структура для преобразование типа аллокатора.

*void deallocate(T\* pointer, std::size\_t)* – функция освобождения выделенной памяти.

*void construct(OTHER\_T\* p, ARGS… arguments)* – функция для вызова конструкторов.

# Структура Node

*T Data* – данные узла.

*std::shared\_ptr<Node> Next* – указатель на следующий элемент очереди.

*friend bool operator== (const Node &left\_value, const Node &right\_value)* – оператор сравнения на равенство двух узлов стека.

*friend bool operator!=* (const Node &left\_value, const Node &right\_value) – оператор сравнения на неравенство двух узлов.

*friend std::ostream &operator<< (std::ostream &output, const Node &node)* – оператор вывода данных узла.

# Структура Deleter

Структура *Deleter* предназначена для освобождения блока с поиощью аллокатора.

*void operator() (void \*pointer)* – оператор для освобождения блока памяти, на который указывает указатель.

# Класс StackIterator

Класс *StackIterator* является вложенным классом класса Stack. Предназначен для использования итераторов в стеке для добавления и удаления элементов стека по указанной позиции в нём.

*std::shared\_ptr<Node<T>> Iterator* – указатель-итератор, с помощью которого можно будет проходить по очереди и удалять или добавлять элементы на указанные позиции.

*void Annul()* – функция, которая приравнивает итератор к nullptr.

*bool IsNull()* – функция проверяющая, является ли итератор пустым указателем.

*bool operator!=(const StackIterator &other\_iterator)* – оператор сравнения двух итераторов на неравенство.

*bool operator==(const StackIterator &other\_iterator*) – оператор сравнения двух итераторов на равенство.

*StackIterator &operator++()* – оператор для перемещения итератора к следующему элементу стека.

*Node &operator\*()* – оператор, возвращающий значение узла, на который указывает итератор.

# Класс Stack

*std::shared\_ptr<Node> TopNode* – указатель на вершину стека.

*int FindSize()* – функция нахождения размера стека.

void Push(const T &element) – функция добавления элемента в стек.

*void Pop()* – функция удаления верхнего элемента из стека.

*T Top()* – функция, возвращающая данные верхнего элемента стека.

*void Insert(StackIterator iter, const T &element)* – функция добавления элемента к стеку по заданному индексу.

*void Erase(StackIterator iter)* – функция удаления элемента из стека по заданному индексу.

*StackIterator begin()* – функция, возвращающая итератор на начало стека.

*StackIterator end()* – функция, возвращающая итератор на конец стека.

# 3. Тесты и результаты их выполнения

Были проведены тесты. Тесты представляют собой введённые данные из файла и результаты выполнения части программы после каждого введённого действия. В качестве данных подаются доступные в меню действия (написаны в пункте 1) и параметры треугольников.

Таблица 2 – test\_01.txt и результаты его выполнения

|  |  |
| --- | --- |
| **Введённые данные** | **Результат** |
| 1 0 0 100 | The figure has been added |
| 1 0 0 65 | The figure has been added |
| 2 7 8 23 2 | The figure has been added |
| 4 6 | Entered the wrong index! |
| 2 1 1 56 1 | The figure has been added |
| 5 | All figures: {(29, -15.1658), (1, 33.3316), (-27, -15.1658)} {(32.5, -18.7639), (0, 37.5278), (-32.5, -18.7639)} {(18.5, 1.36047), (7, 21.2791), (-4.5, 1.36047)} {(50, -28.8675), (0, 57.735), (-50, -28.8675)} |
| 4 2 | The figure has been removed. |
| 3 | The figure has been removed. |
| 5 | All figures:  {(18.5, 1.36047), (7, 21.2791), (-4.5, 1.36047)} {(50, -28.8675), (0, 57.735), (-50, -28.8675)} |
| 6 | The program has been completed. |

Таблица 3 – test\_02.txt и результаты его выполнения

|  |  |
| --- | --- |
| **Введённые данные** | **Результат** |
| 3 | Error: Stack is empty! |
| 2 8 | Error: Entered the wrong index! |
| 1 98 4432 654 | The figure has been added |
| 1 4234 7657 543 | The figure has been added |
| 5 | All figures:  {(4505.5, 7500.25), (4234, 7970.5), (3962.5, 7500.25)} {(425, 4243.21), (98, 4809.59), (-229, 4243.21)} |
| 2 0 0 10000 2 | The figure has been added |
| 2 3 546 786 453 | Error: Entered the wrong index! |
| 5 | All figures:  {(4505.5, 7500.25), (4234, 7970.5), (3962.5, 7500.25)} {(5000, -2886.75), (0, 5773.5), (-5000, -2886.75)} {(425, 4243.21), (98, 4809.59), (-229, 4243.21)} |
| 4 2 | The figure has been removed. |
| 1 0 0 10 | The figure has been added |
| 1 3 3 100 | The figure has been added |
| 3 | Enter an action: The figure has been removed |
| 5 | All figures:  {(5, -2.88675), (0, 5.7735), (-5, -2.88675)} {(4505.5, 7500.25), (4234, 7970.5), (3962.5, 7500.25)} {(425, 4243.21), (98, 4809.59), (-229, 4243.21)} |
| 6 | The program has been completed. |

Таблица 4 – test\_03.txt и результаты его выполнения

|  |  |
| --- | --- |
| **Введённые данные** | **Результат** |
| 1 0 0 75 | The figure has been added |
| 1 -32 -43 32 | The figure has been added |
| 2 -70 65 30 2 | The figure has been added |
| 2 -100 -100 25 3 | The figure has been added |
| 5 | All figures: {(-16, -52.2376), (-32, -24.5248), (-48, -52.2376)} {(-55, 56.3397), (-70, 82.3205), (-85, 56.3397)} {(-87.5, -107.217), (-100, -85.5662), (-112.5, -107.217)} {(37.5, -21.6506), (0, 43.3013), (-37.5, -21.6506)} |
| 4 1 | The figure has been removed. |
| 5 | All figures: {(-55, 56.3397), (-70, 82.3205), (-85, 56.3397)} {(-87.5, -107.217), (-100, -85.5662), (-112.5, -107.217)} {(37.5, -21.6506), (0, 43.3013), (-37.5, -21.6506)} |
| 2 -86 -89 40 3 | The figure has been added |
| 5 | {(-55, 56.3397), (-70, 82.3205), (-85, 56.3397)} {(-87.5, -107.217), (-100, -85.5662), (-112.5, -107.217)} {(-66, -100.547), (-86, -65.906), (-106, -100.547)} {(37.5, -21.6506), (0, 43.3013), (-37.5, -21.6506)} |
| 2 65 89 58 3 | The figure has been added |
| 5 | All figures:  {(-55, 56.3397), (-70, 82.3205), (-85, 56.3397)} {(-87.5, -107.217), (-100, -85.5662), (-112.5, -107.217)} {(94, 72.2568), (65, 122.486), (36, 72.2568)} {(-66, -100.547), (-86, -65.906), (-106, -100.547)} {(37.5, -21.6506), (0, 43.3013), (-37.5, -21.6506)} |
| 3 | The figure has been removed |
| 5 | All figures:  {(-87.5, -107.217), (-100, -85.5662), (-112.5, -107.217)} {(94, 72.2568), (65, 122.486), (36, 72.2568)} {(-66, -100.547), (-86, -65.906), (-106, -100.547)} {(37.5, -21.6506), (0, 43.3013), (-37.5, -21.6506)} |
| 6 | The program has been completed. |

# 5. Листинг программы

**CMakeLists.txt**

cmake\_minimum\_required (VERSION 3.2)

project (OOP)

add\_executable (oop\_exercise\_06 main.cpp menu.cpp)

set\_target\_properties(oop\_exercise\_06 PROPERTIES CXX\_STANDART 17

CXX\_STANDART\_REQUIRED ON)

**triangle.hpp**

#pragma once

#include <iostream>

#include <utility>

#include <cmath>

template <class T>

struct Triangle {

    using vertex\_t = std::pair<T,T>;

    vertex\_t Center;

    T Side;

    Triangle(): Center(), Side() {}

    Triangle(const vertex\_t &center, const T &side): Center(center), Side(side) {}

    // Assignment operator

    Triangle& operator=(const Triangle& other) {

        Center = other.Center;

        Side = other.Side;

        return \*this;

    }

};

// Operator of printing triangle coordinates

template <class T>

std::ostream& operator << (std::ostream& cout, const Triangle<T> &object) {

    double height = object.Side \* sqrt(3) / 2.0;

    cout << "{";

    cout << "(" << object.Center.first + object.Side / 2.0 << ", " <<

object.Center.second - height / 3.0 << "), ";

    cout << "(" << object.Center.first << ", " << object.Center.second +

2 \* height / 3.0 << "), ";

    cout << "(" << object.Center.first - object.Side / 2.0 << ", " << object.Center.second - height / 3.0 << ")";

    cout << "}";

    return cout;

}

**allocator.hpp**

#pragma once

#include <iostream>

#include <vector>

template<class T, size\_t BLOCK\_SIZE>

class allocator {

private:

    T\* Buffer;

    std::vector<T\*> Vector;

public:

    using allocator\_type = allocator;

    using value\_type = T;

    using pointer = T\*;

    using const\_pointer = const T\*;

    using size\_type = std::size\_t;

    allocator(): Vector(), Buffer(nullptr) {

        static\_assert(BLOCK\_SIZE > 0);

    }

    explicit allocator (const allocator<T, BLOCK\_SIZE> &another\_allocator):

allocator() {

        Buffer = new T[BLOCK\_SIZE];

        for (std::size\_t i = 0; i < BLOCK\_SIZE; ++i) {

            Buffer[i] = another\_allocator.Buffer[i];

            Vector.push\_back(&Buffer[i]);

        }

    }

    T \*allocate(const std::size\_t &size) {

        if (Buffer == nullptr) {

            Buffer = new T[BLOCK\_SIZE];

            for (int i = BLOCK\_SIZE - 1; i >= 0; i--) {

                Vector.push\_back(&Buffer[i]);

            }

        }

        if (Vector.size() < size) {

            throw("There is no free blocks to allocate memory!

Please, remove an item from the collection to add a new item.");

        }

        else {

            T\* pointer = Vector.back();

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

                Vector.pop\_back();

            }

            return pointer;

        }

    }

    template<class U>

    struct rebind {

        using other = allocator<U, BLOCK\_SIZE>;

    };

    void deallocate(T\* pointer, std::size\_t) {

        ;

    }

    template<class OTHER\_T, class... ARGS>

    void construct(OTHER\_T\* p, ARGS... arguments) {

        \*p = OTHER\_T(std::forward<ARGS>(arguments)...);

    }

    ~allocator() {

        delete [] Buffer;

    }

};

**stack.hpp**

#pragma once

#include <iostream>

#include <memory>

#include <exception>

#include "allocator.hpp"

template <class T, class ALLOCATOR>

class Stack {

private:

    struct Node;

    using allocator\_type = typename ALLOCATOR::template rebind<Node>::other;

    struct Deleter {

        allocator\_type node\_deleter;

        Deleter(): node\_deleter() {};

        Deleter(allocator\_type \*another\_deleter): node\_deleter(another\_deleter) {}

        void operator() (void \*pointer) {

            node\_deleter.deallocate((Node\*)pointer, 1);

        }

    };

    struct Node {

        T Data;

        std::shared\_ptr<Node> Next;

        Node() noexcept: Data(), Next(nullptr) {};

        explicit Node(const T &element) noexcept: Data(element), Next(nullptr)

{}

        friend bool operator == (const Node &left\_value,

const Node &right\_value) {

            return &left\_value.Data == &right\_value.Data;

        }

        friend bool operator != (const Node &left\_value,

const Node &right\_value) {

            return &left\_value.Data != &right\_value.Data;

        }

        friend std::ostream &operator<< (std::ostream & output,

const Node &node) {

            output << node.Data;

            return output;

        }

    };

public:

    class StackIterator {

    private:

        std::shared\_ptr<Node> Iterator;

    public:

        using iterator\_category = std::forward\_iterator\_tag;

        using difference\_type = std::ptrdiff\_t;

        using value\_type = T;

        using pointer = T\*;

        using reference = T&;

        StackIterator(): Iterator(nullptr) {}

        explicit StackIterator(const std::shared\_ptr<Node> &another\_iter):

Iterator(another\_iter) {}

        void Annul() {

            Iterator = nullptr;

        }

        bool IsNull() {

            return Iterator == nullptr;

        }

        bool operator!=(const StackIterator &other\_iterator) {

            return &other\_iterator.Iterator->Data != &this->Iterator->Data;

        }

        bool operator==(const StackIterator &other\_iterator) {

            return &other\_iterator.Iterator->Data == &this->Iterator->Data;

        }

        StackIterator &operator++() {

            if (this->Iterator != nullptr) {

                this->Iterator = this->Iterator->Next;

                return \*this;

            }

            else {

                throw("Iterator points to nullptr!");

            }

        }

        Node &operator\*() {

            return \*Iterator;

        }

    };

private:

    std::shared\_ptr<Node> TopNode;

    int Size = 0;

    Deleter Stack\_deleter;

public:

    Stack() noexcept: TopNode() {}

    int FindSize() {

        return Size;

    }

    void Push(const T &element) {

        Node \*new\_node = Stack\_deleter.node\_deleter.allocate(sizeof(Node));

        Stack\_deleter.node\_deleter.construct(new\_node, element);

        std::shared\_ptr<Node> new\_node\_shared(new\_node, Stack\_deleter);

        new\_node\_shared->Next = TopNode;

        TopNode = new\_node\_shared;

        Size++;

    }

    void Pop() {

        if (TopNode) {

            TopNode = TopNode->Next;

            Size--;

        }

        else {

            throw ("Stack is empty!");

        }

    }

    T Top() {

        if (TopNode) {

            return TopNode->Data;

        }

        else {

            throw ("Stack is empty!");

        }

    }

    void Insert(StackIterator iter, const T &element) {

        Node\* new\_node = Stack\_deleter.node\_deleter.allocate(sizeof(Node));

        Stack\_deleter.node\_deleter.construct(new\_node, element);

        std::shared\_ptr<Node> new\_node\_shared(new\_node, Stack\_deleter);

        if (TopNode) {

            if (\*iter == \*TopNode) {

                new\_node\_shared->Next = TopNode;

                TopNode = new\_node\_shared;

                iter.Annul();

                return;

            }

            std::shared\_ptr<Node> previous\_node = TopNode;

            while (\*previous\_node->Next != \*iter) {

                previous\_node = previous\_node->Next;

            }

            if (iter.IsNull()) {

                previous\_node->Next = new\_node\_shared;

            }

            else {

                new\_node\_shared->Next = previous\_node->Next;

                previous\_node->Next = new\_node\_shared;

            }

        }

        else {

            TopNode = new\_node\_shared;

        }

        Size++;

        iter.Annul();

    }

    void Erase(StackIterator iter) {

        if (iter.IsNull()) {

        }

        else {

            if (\*iter == \*TopNode) {

                TopNode = TopNode->Next;

            }

            else {

                std::shared\_ptr<Node> previous\_node = TopNode;

                while (\*previous\_node->Next != \*iter) {

                    previous\_node = previous\_node->Next;

                }

                previous\_node->Next = previous\_node->Next->Next;

                (\*iter).Next = nullptr;

            }

            iter.Annul();

        }

        Size--;

    }

    StackIterator begin() {

        return StackIterator(TopNode);

    }

    StackIterator end() {

        return StackIterator(nullptr);

    }

    ~Stack() {}

};

**menu.hpp**

#pragma once

int menu();

**menu.cpp**

#include <iostream>

#include <algorithm>

#include <exception>

#include "stack.hpp"

#include "triangle.hpp"

int menu() {

    const std::size\_t BLOCK\_SIZE = 256;

    Stack<Triangle<int>, allocator<Triangle<int>, BLOCK\_SIZE>> stack;

    while(1) {

        std::cout << "Menu:\n";

        std::cout << "1 - Add figure in collection as stack\n";

        std::cout << "2 - Add figure in collection by index\n";

        std::cout << "3 - Delete figure from the collection as from stack\n";

        std::cout << "4 - Delete figure from the collection by index\n";

        std::cout << "5 - Print figures from the collection\n";

        std::cout << "6 - End the program\n";

        int variant = 0;

        std::cout << "Enter an action: ";

        std::cin >> variant;

        switch (variant) {

        case 1:

        {

            int x,y;

            int side;

            std::cout << "Write the center coordinates: ";

            std::cin >> x >> y;

            std::cout << "Write size of the triangle: ";

            std::cin >> side;

            try {

                stack.Push(Triangle<int>(std::pair<int,int> {x, y}, side));

                std::cout << "The figure has been added\n";

            }

            catch (const char\* error) {

                std::cout << "Error: " << error << "\n";

            }

            break;

        }

        case 2:

        {

            int x,y;

            int side;

            int index;

            std::cout << "Write the center coordinates: ";

            std::cin >> x >> y;

            std::cout << "Write size of the triangle: ";

            std::cin >> side;

            std::cout << "Write an index: ";

            std::cin >> index;

            try {

                if (stack.FindSize() + 1 < index) {

                    throw ("Entered the wrong index!");

                }

                else {

                    auto iterator = stack.begin();

                    while(--index) {

                        ++iterator;

                    }

                    stack.Insert(iterator, Triangle<int>(std::pair<int,int>

{x, y}, side));

                    std::cout << "The figure has been added\n";

                }

            }

            catch (const char\* error){

                std::cout << "Error: " << error << "\n";

            }

            break;

        }

        case 3:

        {

            try {

                stack.Pop();

                std::cout << "The figure has been removed\n";

            }

            catch (const char\* error) {

                std::cout << "Error: " << error << "\n";

            }

            break;

        }

        case 4:

        {

            int index;

            std::cout << "Enter the position of the element to remove: ";

            std::cin >> index;

            try {

                if (stack.FindSize() < index) {

                    throw ("Entered the wrong index!\n");

                }

                auto iterator = stack.begin();

                while (--index) {

                    ++iterator;

                }

                stack.Erase(iterator);

                std::cout << "The figure has been removed.\n";

            }

            catch(const char\* error) {

                std::cout << "Error: " << error;

            }

            break;

        }

        case 5:

        {

            auto print\_stack = [](const auto & figure) {

                std::cout << figure << "\n";

            };

            std::cout << "All figures:\n";

            std::for\_each(stack.begin(), stack.end(), print\_stack);

            break;

        }

        case 6:

        {

            std::cout << "The program has been completed.\n";

            return 0;

            break;

        }

        default:

            std::cout << "Entered invalid action!\n";

            break;

        }

        std::cout << "\n";

    }

    return 0;

}

**main.cpp**

#include <iostream>

#include "menu.hpp"

int main() {

   menu();

}

Ссылка на github репозиторий: <https://github.com/HarryLiker/oop_exercise_06.git>

# 6. Выводы

Была написана программа, в которой реализован шаблон класса треугольник. Параметром шаблона является скалярный тип, задающий тип данных для оси координат. Для хранения треугольников был реализован стек с помощью умных указателей и итераторов. Программа может проходить по элементам в коллекции с помощью итераторов, удалять и добавлять элементы. Добавление и удаление элемента из коллекции выполнялось с помощью аллокатора, который позволял выделять и освобождать память для работы с коллекцией. Количество блоков памяти для выделения было фиксированным. Программа может выводить все элементы стека с помощью std::for\_each(). Были также написаны исключения для случаев ввода неверного индекса, добавления нового элемента, когда нет свободных блоков для выделения памяти, пустоты стека.

Выполняя лабораторную работу, были получены навыки работы с умными указателями std::shared\_ptr, работы с итераторами с помощью forward\_iterator, которые можно использовать в коллекции. Были получены навыки для написания собственного аллокатора, его основных функций и его использования со своими структурами, а также со стандартными контейнерами. Были получены знания об обработке исключений. Были приобретены навыки работы с лямбда-выражениями.

# Список используемых источников

1. Р. Лафоре «Объектно-ориентированное программирование в C++» – Москва: «Питер», 2004.

2. Информация об умном указателе std::shared\_ptr – [Электронный ресурс]. – URL: <https://ravesli.com/urok-194-std-shared_ptr/> (дата обращения: 02.12.2020).

4. Информация о перегрузке операторов ввода /вывода – [Электронный ресурс]. – URL: <https://ravesli.com/urok-133-peregruzka-operatorov-vvoda-i-vyvoda/>

(дата обращения: 02.12.2020).

5. Информация о шаблоне std::pair – [Электронный ресурс]. – URL: <https://en.cppreference.com/w/cpp/utility/pair> (дата обращения: 02.12.2020).

6. Информация о спецификаторе auto – [Электронный ресурс]. – URL: <https://ru.cppreference.com/w/cpp/language/auto> (дата обращения: 02.12.2020).

9. Информация о работе с итераторами – [Электронный ресурс]. – URL: <https://ravesli.com/urok-198-iteratory-stl/> (дата обращения: 03.12.2020).

10. Информация о работе с итераторами – [Электронный ресурс]. – URL: <http://www.realcoding.net/articles/iteratory.html> (дата обращения: 03.12.2020).

11. Информация о работе с итераторами – [Электронный ресурс]. – URL: <https://codelessons.ru/cplusplus/iteratory-v-c-vvedenie.html> (дата обращения: 03.12.2020).

12. Информация о работе с итераторами – [Электронный ресурс]. – URL: <http://www.realcoding.net/articles/iteratory.html> (дата обращения: 03.12.2020).

16. Информация об обработке исключений – [Электронный ресурс]. – URL: <https://ravesli.com/urok-182-obrabotka-isklyuchenij/> (дата обращения: 03.12.2020).

17. Информация об алокаторах памяти – [Электронный ресурс]. – URL: <https://habr.com/ru/post/505632/> (дата обращения: 04.12.2020).

18. Информация об альтернативных аллокаторах памяти – [Электронный ресурс]. – URL: <https://habr.com/ru/post/274827/> (дата обращения: 04.12.2020).

19. Типы, необходимые для создания собственного аллокатора ­– [Электронный ресурс]. – URL: <http://www.cplusplus.com/reference/memory/allocator_traits/>

(дата обращения: 04.12.2020).