Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет: «Информационных технологий и прикладной математики»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 5**

**Тема: Основы работы с коллекциями: итераторы**

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

Москва 2020

Оглавление

[1. Постановка задачи 3](#_Toc57023256)

[2. Описание программы 4](#_Toc57023257)

[3. Тесты и результаты их выполнения 7](#_Toc57023262)

[5. Листинг программы 10](#_Toc57023263)

[6. Выводы 18](#_Toc57023264)

[Список используемых источников 19](#_Toc57023265)

# 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.

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. Реализовать forward\_iterator по коллекции;
4. Коллекция должна возвращать итераторы begin() и end();
5. Коллекция должна содержать метод удаления из позиции итератора erase(iterator);
6. При выполнении недопустимых операций (например выход за границы коллекции или удаление не существующего элемента) необходимо генерировать исключения;
7. Итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if).
8. Коллекция должна содержать метод доступа:

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

1. Реализовать программу, которая:

* Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию;
* Позволяет удалять элемент из коллекции по номеру элемента;
* Выводит на экран введённые фигуры с помощью std::for\_each;
* Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if).

Таблица 1 – вариант задания

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Фигура** | **Контейнер** |
| 17 | Треугольник | Очередь |

# 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 конец очереди новую фигуру.
2. Добавить в очередь фигуру на заданную позицию.
3. Удалить фигуру из начала очереди.
4. Удалить из очереди фигуру, находящуюся на заданной позиции.
5. Вывести координаты фигур, находящихся в очереди.
6. Вывести количество фигур, находящихся в очереди, у которых площадь меньше заданной.
7. Завершить программу.

Коллекция (очередь) 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.

Был реализован forward\_terator – последовательный итератор по коллекции.

Коллекция содержит итераторы begin() и end(), которые указывают на начало и конец очереди.

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. Также было добавлено std::runtime\_error() для выброса в случае различных ошибок во время выполнения программы.

Итератор совместим со стандартными алгоритмами. С помощью std::count\_if() программа может находить количество треугольников в коллекции, площадь которых меньше указанной.

Также было добавлено несколько лямбда-выражений для упрощения кода.

# Структура Triangle

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

*T Side* – сторона треугольника.

*double Square() const* – функция нахождения площади 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.

# Структура Node

*T Data* – данные узла.

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

friend void operator++<>(std::shared\_ptr<Node<T>> &node) – оператор перемещения элемента к следующему узлу.

*friend bool operator==<>(const Node<T> &left\_value, const Node<T> &right\_value)* – оператор сравнения двух узлов.

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

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

# Класс Queue

*size\_t Size* – размер очереди (количество элементов, находящихся в ней).

*std::shared\_ptr<Node<T>> First* – указатель на первый элемент очереди.

*std::shared\_ptr<Node<T>> Last* – указатель на последний элемент очереди.

*void Push(const T &element)* – функция, добавляющая в конец очереди новый элемент.

*void Pop()* – функция, удаляющая из начала очереди элемент.

*T Top()* – функция, возвращающая значения, находящиеся в первом элементе очереди.

*void Insert(Queue<T>::QueueIterator &iter, const T &elem)* – функция добавления нового элемента в очередь на указанную позицию. Выполняется с помощью итераторов.

*void Erase(QueueIterator &iter)* – функция удаления элемента из очереди, стоящего на указанной позиции в нём. Выполняется с помощью итераторов.

*QueueIterator begin()* – возвращает итератор на начало коллекции.

*QueueIterator end()* – возвращает итератор на конец коллекции.

*size\_t FindSize()* – возвращает размер коллекции.

# Класс QueueIterator

Класс QueueIterator является вложенным классом класса Queue.

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

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

*friend void operator++(QueueIterator &iter)* – оператор для перемещения итератора к следующему элементу.

*friend bool operator!=(const Queue<T>::QueueIterator &left\_value, const Queue<T>::QueueIterator &right\_value)* – оператор сравнения двух элементов. Возвращает true, если два итератора указывают на разные элементы.

*std::ostream &operator<<(std::ostream &out, const QueueIterator &iter)* – оператор вывода данных итератора.

*Node<T> &operator\*()* – возвращает значение узла, на которое указывает итератор

# 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 5 | The figure has been added |
| 1 87 -90 76 | The figure has been added |
| 5 | All figures:  {(2.5, -1.44338), (0, 2.88675), (-2.5, -1.44338)} {(125, -111.939), (87, -46.1214), (49, -111.939)} |
| 2 1 98 98 1 | The figure has been added |
| 2 4 10 43 4 | The figure has been added |
| 5 | All figures: {(50, 69.7098), (1, 154.58), (-48, 69.7098)} {(25.5, -2.41303), (4, 34.8261), (-17.5, -2.41303)} {(2.5, -1.44338), (0, 2.88675), (-2.5, -1.44338)} {(125, -111.939), (87, -46.1214), (49, -111.939)} |
| 4 3 | The figure has been removed |
| 5 | All figures: {(50, 69.7098), (1, 154.58), (-48, 69.7098)} {(25.5, -2.41303), (4, 34.8261), (-17.5, -2.41303)} {(125, -111.939), (87, -46.1214), (49, -111.939)} |
| 3 | The figure has been removed |
| 6 1000 | Triangles with square less than the specified area: 1 |
| 5 | All figures: {(25.5, -2.41303), (4, 34.8261), (-17.5, -2.41303)} {(125, -111.939), (87, -46.1214), (49, -111.939)} |
| 7 | Enter an action: The program has been completed. |

Таблица 3 – test\_02.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:  {(37.5, -21.6506), (0, 43.3013), (-37.5, -21.6506)} {(-55, 56.3397), (-70, 82.3205), (-85, 56.3397)} {(-87.5, -107.217), (-100, -85.5662), (-112.5, -107.217)} {(-16, -52.2376), (-32, -24.5248), (-48, -52.2376)} |
| 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)} {(-16, -52.2376), (-32, -24.5248), (-48, -52.2376)} |
| 2 -86 -89 40 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)} {(-66, -100.547), (-86, -65.906), (-106, -100.547)} {(-16, -52.2376), (-32, -24.5248), (-48, -52.2376)} |
| 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)} {(-16, -52.2376), (-32, -24.5248), (-48, -52.2376)} |
| 6 500 | Triangles with square less than the specified area: 3 |
| 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)} {(-16, -52.2376), (-32, -24.5248), (-48, -52.2376)} |
| 7 | The program has been completed. |

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

|  |  |
| --- | --- |
| **Введённые данные** | **Результат** |
| 2 0 0 67 1 | The figure has been added |
| 2 0 8 69 2 | The figure has been added |
| 5 | {(33.5, -19.3412), (0, 38.6825), (-33.5, -19.3412)} {(34.5, -11.9186), (0, 47.8372), (-34.5, -11.9186)} |
| 4 1 | The figure has been removed |
| 3 | The figure has been removed |
| 5 | All figures: |
| 1 87 83 90 | The figure has been added |
| 1 32 54 67 | The figure has been added |
| 1 43 6 43 | The figure has been added |
| 5 | {(132, 57.0192), (87, 134.962), (42, 57.0192)} {(65.5, 34.6588), (32, 92.6825), (-1.5, 34.6588)} {(64.5, -6.41303), (43, 30.8261), (21.5, -6.41303)} |
| 2 3 0 98 2 | The figure has been added |
| 6 2000 | Triangles with square less than the specified area: 2 |
| 7 | The program has been completed. |

# 5. Листинг программы

**CMakeLists.txt**

cmake\_minimum\_required (VERSION 3.2)

project (OOP)

add\_executable (oop\_exercise\_05 main.cpp menu.cpp)

set\_target\_properties(oop\_exercise\_05 PROPERTIES CXX\_STANDART 17 CXX\_STANDART\_REQUIRED ON)

**triangle.hpp**

#pragma once

#include <utility>

#include <cmath>

template <class T>

struct Triangle {

    using vertex\_t = std::pair<T,T>;

    vertex\_t Center;

    T Side;

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

    double Square() const {

        return Side \* Side \* sqrt(3) / 4.0;

    }

    // 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;

}

**queue.hpp**

#include <iostream>

#include <memory>

#include <iterator>

#include <exception>

template <class T>

struct Node;

template <class T>

void operator++(std::shared\_ptr<Node<T>> &current\_node) {

    if (current\_node) {

        current\_node = current\_node->Next;

    }

    else {

        throw std::out\_of\_range("Iterator has already nullptr\n");

    }

}

template <class T>

bool operator!=(const Node<T> &left\_value, const Node<T> &right\_value) {

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

}

template <class T>

bool operator==(const Node<T> &left\_value, const Node<T> &right\_value) {

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

}

template <class T>

std::ostream &operator<<(std::ostream &cout, const Node<T> &node) {

    cout << node.Data;

    return cout;

}

template <class T>

struct Node {

    T Data;

    std::shared\_ptr<Node> Next;

    Node(): Next(nullptr) {}

    Node(const T &data): Data(data), Next(nullptr) {}

    friend void operator++<>(std::shared\_ptr<Node<T>> &node);

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

&right\_value);

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

&right\_value);

    friend std::ostream &operator<<<>(std::ostream &cout, const Node<T> &node);

};

template <class T>

class Queue {

private:

    size\_t Size = 0;

    std::shared\_ptr<Node<T>> First;

    std::shared\_ptr<Node<T>> Last;

public:

    class QueueIterator {

    private:

        std::shared\_ptr<Node<T>> 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&;

        QueueIterator(): iterator(nullptr) {}

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

iterator(another\_iter) {}

        bool IsNull() {

            return iterator == nullptr;

        }

        friend void operator++(QueueIterator &iter) {

            ++iter.iterator;

        }

        friend bool operator!=(const Queue<T>::QueueIterator &left\_value, const Queue<T>::QueueIterator &right\_value) {

            return left\_value.iterator != right\_value.iterator;

        }

        friend std::ostream &operator<<(std::ostream &out, const QueueIterator

&iter) {

            out << \*iter.iterator;

            return out;

        }

        Node<T> &operator\*() {

            return \*iterator;

        }

    };

    Queue() noexcept: First(nullptr) {}

    void Push(const T &element) {

        if (Size == 0) {

            Node<T> \*new\_node = new Node<T>(element);

            std::shared\_ptr<Node<T>> new\_pointer{new\_node};

            Last = new\_pointer;

            First = Last;

        }

        else if (Size == 1) {

            Node<T> \*new\_node = new Node<T>(element);

            std::shared\_ptr<Node<T>> new\_pointer{new\_node};

            First->Next = Last;

            Last->Next = new\_pointer;

            Last = new\_pointer;

        }

        else {

            Node<T> \*new\_node = new Node<T>(element);

            std::shared\_ptr<Node<T>> new\_pointer{new\_node};

            Last->Next = new\_pointer;

            Last = new\_pointer;

        }

        ++Size;

    }

    void Pop() {

        if (First) {

            First = First->Next;

            --Size;

        }

        else {

            throw std::runtime\_error("Queue is empty");

        }

    }

    T Top () {

        if (First) {

            return First->Data;

        }

        else {

            throw std::runtime\_error("First is nullptr!\n");

        }

    }

    void Insert(Queue<T>::QueueIterator &iter, const T &elem) {

        std::unique\_ptr<Node<T>> new\_node {new Node<T>(elem)};

        std::shared\_ptr<Node<T>> new\_pointer = std::move(new\_node);

        std::shared\_ptr<Node<T>> next\_pointer = First;

        if (\*iter == \*First) {

            new\_pointer->Next = First;

            First = new\_pointer;

        }

        else if (next\_pointer) {

            if (!iter.IsNull()) {

                while(\*next\_pointer->Next != \*iter) {

                    ++next\_pointer;

                }

            }

            else {

                while (new\_pointer->Next != nullptr) {

                    ++next\_pointer;

                }

            }

            if (next\_pointer->Next == nullptr) {

                next\_pointer->Next = new\_pointer;

                Last = new\_pointer;

            }

            else {

                new\_pointer->Next = next\_pointer->Next;

                next\_pointer->Next = new\_pointer;

            }

        }

        else {

            First = new\_pointer;

        }

        ++Size;

    }

    void Erase(QueueIterator &iter) {

        if (iter.IsNull()) {

            throw std::runtime\_error("Iterator is in nullptr state!\n");

        }

        else {

            if (\*iter == \*First) {

                First = First->Next;

            }

            else {

                std::shared\_ptr<Node<T>> next\_ptr = First;

                while (\*next\_ptr->Next != \*iter) {

                    ++next\_ptr;

                }

                next\_ptr->Next = next\_ptr->Next->Next;

            }

            --Size;

        }

    }

    QueueIterator begin() {

        return QueueIterator(First);

    }

    QueueIterator end() {

        return QueueIterator(Last->Next);

    }

    size\_t FindSize() {

        return Size;

    }

    ~Queue() {}

};

**menu.hpp**

#pragma once

int menu();

**menu.cpp**

#include <iostream>

#include <memory>

#include <algorithm>

#include "menu.hpp"

#include "triangle.hpp"

#include "queue.hpp"

int menu() {

    Queue<Triangle<int>> queue;

    while (1) {

        int variant;

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

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

        std::cout <<"3 - Delete an element from the collection as from queue\n";

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

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

        std::cout << "6 - Print figures, with less square\n";

        std::cout << "7 - End the program\n";

        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 side of triangle: ";

            std::cin >> side;

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

            std::cout << "The figure has been added\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 side of triangle: ";

            std::cin >> side;

            std::cout << "Enter the position of the new item in the queue: ";

            std::cin >> index;

            try {

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

                    // if the index is outside the collection or negative

                    throw std::invalid\_argument("Entered the wrong index!\n");

                }

                else {

                    auto iterator = queue.begin();

                    while (--index) {

                        ++iterator;

                    }

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

{x, y}, side));

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

                }

            }

            // If an excenpion occurs

            catch (std::invalid\_argument &argument) {

                std::terminate();

            }

            break;

        }

        case 3:

        {

            queue.Pop();

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

            break;

        }

        case 4:

        {

            int index;

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

            std::cin >> index;

            try {

                auto iterator = queue.begin();

                while (--index) {

                    ++iterator;

                }

                queue.Erase(iterator);

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

            }

            catch (std::runtime\_error &error) {

                std::cout << error.what() << "\n";

            }

            break;

        }

        case 5:

        {

            auto print\_queue = [](const Node<Triangle<int>> &figure) {

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

            };

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

            std::for\_each(queue.begin(), queue.end(), print\_queue);

            break;

        }

        case 6:

        {

            double minimal\_suare;

            std::cout << "Enter the minimal square: ";

            std::cin >> minimal\_suare;

            auto figures\_less\_squre = [&minimal\_suare](const Node<Triangle<int>> &figure) {

                return figure.Data.Square() < minimal\_suare;

            };

            std::cout << "Triangles with square less than the specified area: " << std::count\_if(queue.begin(), queue.end(), figures\_less\_squre) << "\n";

            break;

        }

        case 7:

        {

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

            return 0;

            break;

        }

        default:

        {

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

            break;

        }

        }

        std::cout << "\n";

    }

}

**main.cpp**

#include <iostream>

#include "menu.hpp"

int main() {

    menu();

}

Ссылка на github репозиторий: https://github.com/HarryLiker/oop\_exercise\_05

# 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 была реализована очередь с помощью умных указателей и итераторов. Программа может проходить по элементам в коллекции с помощью итераторов, удалять и добавлять элементы. Были также написаны исключения для случаев ввода неверного индекса, выхода за предел коллекции и удаления несуществующего элемента.

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 знания об обработке исключений. Были навыки работы с люмбда-выражениями. Были получены знания об std::count\_if(), который позволяет программе проверять условие выполнения для итераторов в заданном диапазоне. Также были получены знание об алгоритме for\_each(). С помощью него программа выводит координаты фигур, находящихся в контейнере.

# 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: 21.11.2020).

3. Информация об умном указателе std::weak\_ptr – [Электронный ресурс]. – URL: <https://ravesli.com/urok-195-std-weak_ptr-tsiklicheskaya-zavisimost-s-std-shared_ptr/> (дата обращения: 21.11.2020).

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

(дата обращения: 21.11.2020).

5. Информация о шаблоне std::pair – [Электронный ресурс]. – URL: <https://en.cppreference.com/w/cpp/utility/pair> (дата обращения: 21.11.2020).

6. Информация о спецификаторе auto – [Электронный ресурс]. – URL: <https://ru.cppreference.com/w/cpp/language/auto> (дата обращения: 21.11.2020).

7. Информация о контейнере очередь – [Электронный ресурс]. – URL: <https://prog-cpp.ru/data-queue/> (дата обращения: 21.11.2020).

8. Информация о контейнере очередь – [Электронный ресурс]. – URL: <https://codelessons.ru/cplusplus/ochered-queue-v-c-realizaciya-i-chto-eto-voobshhe-takoe.html> (дата обращения: 21.11.2020).

9. Информация о работе с итераторами – [Электронный ресурс]. – URL: <https://ravesli.com/urok-198-iteratory-stl/> (дата обращения: 21.11.2020).

10. Информация о работе с итераторами – [Электронный ресурс]. – URL: <http://www.realcoding.net/articles/iteratory.html> (дата обращения: 21.11.2020).

11. Информация о работе с итераторами – [Электронный ресурс]. – URL: <https://codelessons.ru/cplusplus/iteratory-v-c-vvedenie.html> (дата обращения: 21.11.2020).

12. Информация о работе с итераторами – [Электронный ресурс]. – URL: <http://www.realcoding.net/articles/iteratory.html> (дата обращения: 21.11.2020).

13. Информация об алгоритме – [Электронный ресурс]. – URL: <https://ru.cppreference.com/w/cpp/algorithm/for_each> (дата обращения: 21.11.2020).

14. Информация о std::move() – [Электронный ресурс]. – URL: <https://tproger.ru/articles/move-semantics-and-rvalue/> (дата обращения: 21.11.2020).

15. Информация о std::move() – [Электронный ресурс]. – URL: <https://ravesli.com/urok-192-std-move/> (дата обращения: 21.11.2020).

16. Информация об обработке исключений – [Электронный ресурс]. – URL: <https://ravesli.com/urok-182-obrabotka-isklyuchenij/> (дата обращения: 21.11.2020).