Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет прикладной математики и информационных технологий

Кафедра вычислительной математики и программирования

Отчет по лабораторным работам

по курсу

Объектно-ориентированное программирование

Студент группы 8О-208Б Фирфаров А.С.

Вариант № 27

Преподаватель Поповкин А.В.

Подпись преподавателя

Оглавление

Лабораторная работа № 1

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Программирование классов на языке C++
- Управление памятью в языке C++
- Изучение базовых понятий ООП.
- Знакомство с классами в C++.
- Знакомство с перегрузкой операторов.
- Знакомство с дружественными функциями.
- Знакомство с операциями ввода-вывода из стандартных библиотек.

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно вариантов задания.

Классы должны удовлетворять следующим правилам:

- Должны иметь общий родительский класс Figure.
- Должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в стандартный поток вывода cout.
- Должный иметь общий виртуальный метод расчета площади фигуры – Square.
- Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin.
- Должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Программа должна позволять вводить фигуру каждого типа с клавиатуры, выводить параметры фигур на экран и их площадь.

Код программы:

Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

class Figure {
public:
    virtual double Square() = 0;
    virtual void   Print() = 0;
    virtual ~Figure() {};
};

#endif
```

Rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Rectangle : public Figure {
public:
    Rectangle(double a = 0, double b = 0);
    Rectangle(std::istream &is);

    double Square() override;
    void   Print() override;

    virtual ~Rectangle();
private:
    double len_rec;
    double height_rec;
};

#endif
```

Rectangle.cpp

```
#include "Rectangle.h"
#include <iostream>
#include <cmath>

Rectangle::Rectangle(double a,double b) : len_rec(a), height_rec(b) {
    std::cout << "Rectangle created: " << std::endl;
}

Rectangle::Rectangle(std::istream &is) {
    std::cout << "Введите длину прямоугольника: ";
    is >> len_rec;
    std::cout << '\n';
    std::cout << "Введите высоту прямоугольника: ";
    is >> height_rec;
    std::cout << '\n';
}
```

```
double Rectangle::Square() {
    return len_rec * height_rec;
}

void Rectangle::Print() {
    std::cout << "Тип: прямоугольник" << std::endl;
    std::cout << "Длина: " << len_rec << std::endl;
    std::cout << "Высота: " << height_rec << std::endl;
}

Rectangle::~Rectangle() {
    std::cout << "Rectangle deleted" << std::endl;
}
```

Rhomb.h

```
#ifndef RHOMB_H
#define RHOMB_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Rhomb : public Figure {
public:
    Rhomb(double a = 0);
    Rhomb(std::istream &is);

    double Square() override;
    void   Print() override;

    virtual ~Rhomb();
private:
    double rhomb_side;
    double rhomb_height;
};

#endif
```

Rhomb.cpp

```
#include "Rhomb.h"
#include <iostream>
#include <cmath>

Rhomb::Rhomb(double a) : rhomb_side(a) {
    std::cout << "Rhomb created: " << std::endl;
}

Rhomb::Rhomb(std::istream &is) {
    std::cout << "Введите сторону ромба: ";
    is >> rhomb_side;
    std::cout << '\n';
    std::cout << "Введите высоту ромба: ";
    is >> rhomb_height;
    std::cout << '\n';
}
```

```
double Rhomb::Square() {
    return rhomb_side * rhomb_height;
}

void Rhomb::Print() {
    std::cout << "Тип: ромб" << std::endl;
    std::cout << "Сторона ромба: " << rhomb_side << std::endl;
    std::cout << "Высота: " << rhomb_height << std::endl;
}

Rhomb::~Rhomb() {
    std::cout << "Rhomb deleted" << std::endl;
}
```

Trapeze.h

```
#ifndef TRAPEZE_H
#define TRAPEZE_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Trapeze : public Figure {
public:
    Trapeze(double a = 0, double b = 0, double c = 0, double d = 0);
    Trapeze(std::istream &is);

    double Square() override;
    void   Print() override;

    virtual ~Trapeze();
private:
    double up_base;
    double low_base;
    double left_side;
    double right_side;
};

#endif
```

Trapeze.cpp

```
#include "Trapeze.h"
#include <iostream>
#include <cmath>

Trapeze::Trapeze(double a,double b,double c,double d) : up_base(a),
low_base(b), left_side(c), right_side(d) {
    std::cout << "Trapeze created: " << std::endl;
}

Trapeze::Trapeze(std::istream &is) {
    std::cout << "Введите верхнее основание трапеции: ";
    is >> up_base;
    std::cout << '\n';
    std::cout << "Введите нижнее основание трапеции: ";
    is >> low_base;
    std::cout << '\n';
    std::cout << "Введите боковые стороны через пробел: ";
    is >> left_side;
```

```
    is >> right_side;
    std::cout << '\n';
}

double Trapeze::Square() {
  double p = (up_base + low_base) / 2.0;
  double temp = ((low_base - up_base) * (low_base - up_base) + left_side *
left_side - right_side * right_side)/(2*(low_base - up_base));
  return p*sqrt(left_side*left_side - temp * temp);
}

void Trapeze::Print() {
    std::cout << "Тип: трапеция" << std::endl;
    std::cout << "Верхнее основание: " <<  up_base << std::endl;
    std::cout << "Нижнее основание: " << low_base << std::endl;
    std::cout << "Левая сторона: " << left_side << std::endl;
    std::cout << "Правая сторона: " << right_side << std::endl;
}

Trapeze::~Trapeze() {
    std::cout << "Trapeze deleted" << std::endl;
}
```

main.cpp

```
#include <cstdlib>
#include <iostream>
#include "Rectangle.h"
#include "Trapeze.h"
#include "Rhomb.h"


int main(int argc, char** argv) {
    char choice;

    puts("----------------------MENU-------------------\n");
    puts("              1 - прямоугольник              \n");
    puts("                2 - трапеция                 \n");
    puts("                  3 - ромб                   \n");
    puts("                  x - выход                  \n");
    puts("-----------------------------------------------\n");

    do {
        std::cout << '\n';
        std::cin >> choice;

        switch(choice) {
            case '1':{
                Figure *ptr = new Rectangle(std::cin);
                ptr->Print();
                std::cout << "Площадь: " <<  ptr->Square() << std::endl;
                delete ptr;
                break;
            }
            case '2':{
                Figure *ptr = new Trapeze(std::cin);
                ptr->Print();
                std::cout << "Площадь: " <<  ptr->Square() << std::endl;
                delete ptr;
                break;
            }
```

```
            case '3':{
                Figure *ptr = new Rhomb(std::cin);
                ptr->Print();
                std::cout << "Площадь: " <<  ptr->Square() << std::endl;
                delete ptr;
                break;
            }
              case 'x':{
                break;
            }
            default:{
                std::cout << "Введите 1,2,3 или x" << std::endl;
                break;
            }
        }
    } while (choice != 'x');

return 0;
}
```

Тестирование:

```
user@lubuntu:~/OOP/lab1$ ./main.exe
----------------------MENU-------------------

              1 - прямоугольник

                2 - трапеция

                  3 - ромб

                  x - выход

-----------------------------------------


1
Введите длину прямоугольника: 5

Введите высоту прямоугольника: 7

Тип: прямоугольник
Длина: 5
Высота: 7
Площадь: 35
Rectangle deleted

2
Введите верхнее основание трапеции: 5

Введите нижнее основание трапеции: 10

Введите боковые стороны через пробел: 4 5

Тип: трапеция
Верхнее основание: 5
Нижнее основание: 10
Левая сторона: 4
Правая сторона: 5
Площадь: 27.4955
Trapeze deleted
```

```
3
Введите сторону ромба: 9

Введите высоту ромба: 8

Тип: ромб
Сторона ромба: 9
Высота: 8
Площадь: 72
Rhomb deleted

x
user@lubuntu:~/OOP/lab1$
```

Лабораторная работа № 2

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Закрепление навыков работы с классами.
- Создание простых динамических структур данных.
- Работа с объектами, передаваемыми «по значению».

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру (колонка фигура 1), согласно вариантов задания (реализованную в ЛР1).

Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<). Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д).
- Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>). Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).
- Классы фигур должны иметь операторы копирования (=).
- Классы фигур должны иметь операторы сравнения с такими же фигурами (==).
- Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера (опеределяется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опеределяется структурой контейнера).
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Различные варианты умных указателей (shared_ptr, weak_ptr).

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Код программы:

TQueue.h

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include "Rectangle.h"
#include "TQueueItem.h"

class TQueue {
public:
    TQueue();

    void push(Rectangle &&rectangle);
    Rectangle pop();
    bool empty();
    friend std::ostream& operator<<(std::ostream& os,const TQueue& queue);
    ~TQueue();
private:

    TQueueItem *head;
    TQueueItem *tail;
};

#endif
```

TQueue.cpp

```
#include "TQueue.h"

TQueue::TQueue() {
    tail = nullptr;
    head = nullptr;
}

std::ostream& operator<<(std::ostream& os, const TQueue& queue) {

    TQueueItem *item = queue.head;

    while(item != nullptr) {

        Rectangle t = item->GetRectangle();
        os << t << std::endl;
        item = item->GetNext();
    }

    return os;
}

void TQueue::push(Rectangle &&rectangle) {
    TQueueItem *other = new TQueueItem(rectangle);
    if (empty()) {
        head = other;
        tail = other;
        other->SetNext(nullptr);
        return;
    }
    tail->SetNext(other);
```

```
    tail = other;
    tail->SetNext(nullptr);
}

bool TQueue::empty() {
    return head == nullptr;
}

Rectangle TQueue::pop() {
    Rectangle result;
    if (!empty()) {
        TQueueItem *old_head = head;
        head = head->GetNext();
        result = old_head->GetRectangle();
        old_head->SetNext(nullptr);
        delete old_head;
        return result;
    }
    else {
        std::cout << "Очередь пуста " << std::endl;
    }
}

TQueue::~TQueue() {
    while (!empty()) {
        TQueueItem *temp = head;
        head = head->GetNext();
        delete temp;
    }
    tail = nullptr;
    head = nullptr;
    std::cout << "Очередь удалена" << std::endl;
}
```

TQueueitem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

#include "Rectangle.h"
class TQueueItem {
public:
    TQueueItem(const Rectangle& rectangle);
    friend std::ostream& operator<<(std::ostream& os, const TQueueItem& obj);

    TQueueItem* SetNext(TQueueItem* next);
    TQueueItem* GetNext();
    Rectangle GetRectangle() const;

    virtual ~TQueueItem();
private:
    Rectangle rectangle;
    TQueueItem *next;
};

#endif
```

TQueueitem.cpp

```
#include "TQueueItem.h"
#include <iostream>

TQueueItem::TQueueItem(const Rectangle& rectangle) {
    this->rectangle = rectangle;
    this->next = nullptr;
}

TQueueItem* TQueueItem::SetNext(TQueueItem* next) {
    this->next = next;
    return this;
}

Rectangle TQueueItem::GetRectangle() const {
    return this->rectangle;
}

TQueueItem* TQueueItem::GetNext() {
    return this->next;
}

std::ostream& operator<<(std::ostream& os, const TQueueItem& obj) {
    os << obj.rectangle << std::endl;
    return os;
}

TQueueItem::~TQueueItem() {
    next = nullptr;
}
```

Rectangle.h

```
friend std::ostream& operator<<(std::ostream& os, Rectangle& obj);
friend std::istream& operator>>(std::istream& is,  Rectangle& obj);
Rectangle& operator=(const Rectangle& right);
bool operator==(const Rectangle& right);
```

Rectangle.cpp

```
Rectangle& Rectangle::operator=(const Rectangle& right) {

    length = right.length;
    height = right.height;

    return *this;
}

bool Rectangle::operator==(const Rectangle& right) {
    if ((length == right.length) && (height == right.height)) {
        return true;
    }
    return false;
}
```

```
std::ostream& operator<<(std::ostream& os, Rectangle& obj) {

    os << "Длина: " << obj.length << ", Высота: " << obj.height << ",
Площадь: " << obj.Square() << std::endl;
    return os;
}

std::istream& operator>>(std::istream& is, Rectangle& obj) {

    do {
        std::cout << "Введите длину: " << std::endl;
        is >> obj.length;
    } while (obj.length < 0);

    do {
        std::cout << "Введите ширину: " << std::endl;
        is >> obj.height;
    } while (obj.height < 0);


    return is;
}
```

main.cpp

```
#include <cstdlib>
#include <iostream>
#include "Rectangle.h"
#include "TQueueItem.h"
#include "TQueue.h"

int main(int argc, char** argv) {

    char choice;
    TQueue queue;

    puts("----------------------MENU--------------------\n");
    puts("            1 - Добавить фигуру             \n");
    puts("             2 - Извлечь фигуру             \n");
    puts("           3 - Просмотреть очередь          \n");
    puts("             4 - Удалить очередь            \n");
    puts("                  x - выход                 \n");
    puts("----------------------------------------------\n");

    do {
        std::cout << '\n';
        std::cin >> choice;

        switch(choice) {
            case '1': {
                Rectangle t;
                std::cin >> t;
                queue.push(std::move(t));
                std::cout << "Добавлено " << std::endl;
                break;
            }
            case '2': {
                Rectangle t;
                t = queue.pop();
                std::cout << t;
```

```
                break;
            }
            case '3': {
                std::cout << queue;
                break;
            }
             case '4': {
                queue.~TQueue();
                break;
            }
              case 'x':{
                break;
            }
            default:{
                std::cout << "Введите 1,2,3,4 или x" << std::endl;
                break;
            }
        }
    } while (choice != 'x');

    return 0;
}
```

Тестирование:

```
user@lubuntu:~/OOP/lab2$ ./main.exe
----------------------MENU-------------------

            1 - Добавить фигуру

             2 - Извлечь фигуру

           3 - Просмотреть очередь

             4 - Удалить очередь

                  x - выход

---------------------------------------------


1
Введите длину:
5
Введите ширину:
7
Добавлено

1
Введите длину:
8
Введите ширину:
9
Добавлено

1
Введите длину:
4
Введите ширину:
5
Добавлено
```

```
1
Введите длину:
12
Введите ширину:
5
Добавлено

1
Введите длину:
4
Введите ширину:
7
Добавлено

3
Длина: 5, Высота: 7, Площадь: 35

Длина: 8, Высота: 9, Площадь: 72

Длина: 4, Высота: 5, Площадь: 20

Длина: 12, Высота: 5, Площадь: 60

Длина: 4, Высота: 7, Площадь: 28


2
Длина: 5, Высота: 7, Площадь: 35

3
Длина: 8, Высота: 9, Площадь: 72

Длина: 4, Высота: 5, Площадь: 20

Длина: 12, Высота: 5, Площадь: 60

Длина: 4, Высота: 7, Площадь: 28


4
Очередь удалена

x
Очередь удалена
user@lubuntu:~/OOP/lab2$
```

Лабораторная работа №3

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Закрепление навыков работы с классами.
- Знакомство с умными указателями.

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **все три** фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1).

Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Класс-контейнер должен соджержать объекты используя std:shared_ptr<…>.
- Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.
- Класс-контейнер должен иметь методы по получению фигуры из контейнера (опередеряется структурой контейнера).
- Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опередеряется структурой контейнера).
- Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).
- Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Объекты «по-значению»

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Код программы:

TQueue.h

```
class TQueue {
public:
    TQueue();

    void push(std::shared_ptr<Figure> &&figure);
    std::shared_ptr<Figure> pop();
    bool empty();
    friend std::ostream& operator<<(std::ostream& os,const TQueue& queue);
    virtual ~TQueue();
private:

    std::shared_ptr<TQueueItem> head;
    std::shared_ptr<TQueueItem> tail;
};
```

TQueue.cpp

```
void TQueue::push(std::shared_ptr<Figure> &&figure) {

    std::shared_ptr<TQueueItem> other(new TQueueItem(figure));
    if (empty()) {
        head = other;
        tail = other;
        std::shared_ptr<TQueueItem> empty = nullptr;
        other->SetNext(empty);
        return;
    }

    tail->SetNext(other);
    tail = other;
    std::shared_ptr<TQueueItem> empty = nullptr;
    tail->SetNext(empty);
}

bool TQueue::empty() {
    return head == nullptr;
}

std::shared_ptr<Figure> TQueue::pop() {

    std::shared_ptr<Figure> result;
    if (!empty()) {
        std::shared_ptr<TQueueItem> old_head = head;
        head = head->GetNext();
        result = old_head->GetFigure();
        std::shared_ptr<TQueueItem> empty = nullptr;
        old_head->SetNext(empty);
        return result;
    }
    else {
        std::cout << "Очередь пуста " << std::endl;
        return nullptr;
    }
}
```

```
TQueue::~TQueue() {
    tail = nullptr;
    head = nullptr;
    std::cout << "Очередь удалена" << std::endl;
}
```

TQueueitem.h

```
class TQueueItem {
public:
    TQueueItem(const std::shared_ptr<Figure>& figure);
    friend std::ostream& operator<<(std::ostream& os, const TQueueItem& obj);

    void SetNext(std::shared_ptr<TQueueItem> &next);
    std::shared_ptr<TQueueItem> GetNext();
    std::shared_ptr<Figure> GetFigure() const;

    virtual ~TQueueItem();
private:
    std::shared_ptr<Figure> figure;
    std::shared_ptr<TQueueItem> next;
};
```

TQueueitem.cpp

```
TQueueItem::TQueueItem(const std::shared_ptr<Figure>& figure) {
    this->figure = figure;
    this->next = nullptr;
}

void TQueueItem::SetNext(std::shared_ptr<TQueueItem>& next) {
    this->next = next;
    return;
}

std::shared_ptr<Figure> TQueueItem::GetFigure() const {
    return this->figure;
}

std::shared_ptr<TQueueItem> TQueueItem::GetNext() {
    return this->next;
}

std::ostream& operator<<(std::ostream& os, const TQueueItem& obj) {
    std::shared_ptr<Figure> figure = obj.figure;
    figure->Print();
    return os;
}
```

Rhomb.h

```
friend std::ostream& operator<<(std::ostream& os, Rhomb& obj);
friend std::istream& operator>>(std::istream& is,  Rhomb& obj);
Rhomb& operator=(const Rhomb& right);
bool operator==(const Rhomb& right);
```

Rhomb.cpp

```
Rhomb& Rhomb::operator=(const Rhomb& right) {

    rhomb_side = right.rhomb_side;
    rhomb_height = right.rhomb_height;

    return *this;
}

bool Rhomb::operator==(const Rhomb& right) {
    if ((rhomb_side == right.rhomb_side) && (rhomb_height ==
right.rhomb_height)) {
        return true;
    }
    return false;
}

std::ostream& operator<<(std::ostream& os, Rhomb& obj) {

    os << "Сторона: " << obj.rhomb_side << ", Высота: " << obj.rhomb_height
<< ", Площадь: " << obj.Square() << std::endl;
    return os;
}

std::istream& operator>>(std::istream& is, Rhomb& obj) {

    do {
        std::cout << "Введите сторону: " << std::endl;
        is >> obj.rhomb_side;
    } while (obj.rhomb_side < 0);

    do {
        std::cout << "Введите высоту: " << std::endl;
        is >> obj.rhomb_height;
    } while (obj.rhomb_height < 0);


    return is;
}
```

Trapeze.h

```
friend std::ostream& operator<<(std::ostream& os, Trapeze& obj);
friend std::istream& operator>>(std::istream& is,  Trapeze& obj);
Trapeze& operator=(const Trapeze& right);
bool operator==(const Trapeze& right);
```

Trapeze.cpp

```
Trapeze& Trapeze::operator=(const Trapeze& right) {

    up_base = right.up_base;
    low_base = right.low_base;
    height = right.height;

    return *this;
}
```

```
bool Trapeze::operator==(const Trapeze& right) {
    if ((up_base == right.up_base) && (low_base == right.low_base) && (height
== right.height)) {
        return true;
    }
    return false;
}

std::ostream& operator<<(std::ostream& os, Trapeze& obj) {

    os << "Верх. осн: " << obj.up_base << ", Нижн. осн: " << obj.low_base <<
", Высота: " << obj.height << ", Площадь: " << obj.Square() << std::endl;
    return os;
}

std::istream& operator>>(std::istream& is, Trapeze& obj) {

    do {
        std::cout << "Введите верх. основание.: " << std::endl;
        is >> obj.up_base;
    } while (obj.up_base < 0);

    do {
        std::cout << "Введите нижн. основание: " << std::endl;
        is >> obj.low_base;
    } while (obj.low_base < 0);

    do {
        std::cout << "Введите высоту: " << std::endl;
        is >> obj.height;
    } while (obj.height < 0);


    return is;
}
```

main.cpp

```
int main(int argc, char** argv) {

    char choice;
    TQueue queue;

    puts("---------------------MENU-------------------\n");
    puts("        r - Добавить прямоугольник          \n");
    puts("             b - Добавить ромб              \n");
    puts("           t - Добавить трапецию            \n");
    puts("             2 - Извлечь фигуру             \n");
    puts("           3 - Просмотреть очередь          \n");
    puts("             4 - Удалить очередь            \n");
    puts("                  x - выход                 \n");
    puts("--------------------------------------------\n");

    do {
        std::cout << '\n';
        std::cin >> choice;

        switch(choice) {
            case 'r': {
                Figure *t = new Rectangle(std::cin);
                queue.push(std::shared_ptr<Figure>(t));
```

```
                std::cout << "Добавлен прямоугольник " << std::endl;
                break;
            }
            case 'b': {
                Figure *t = new Rhomb(std::cin);
                queue.push(std::shared_ptr<Figure>(t));
                std::cout << "Добавлен ромб " << std::endl;
                break;
            }
            case 't': {
                Figure *t = new Trapeze(std::cin);
                queue.push(std::shared_ptr<Figure>(t));
                std::cout << "Добавлена трапеция " << std::endl;
                break;
            }
            case '2': {
                std::shared_ptr<Figure> t;
                t = queue.pop();
                if (t == nullptr) {
                    break;
                }
                t->Print();
                break;
            }
            case '3': {
                std::cout << queue;
                break;
            }
            case '4': {
                queue.~TQueue();
                break;
            }
              case 'x':{
                break;
            }
            default:{
                std::cout << "Нет варианта" << std::endl;
                break;
            }
        }
    } while (choice != 'x');

    return 0;
}
```

Тестирование

```
user@lubuntu:~/OOP/lab3$ ./main.exe
---------------------MENU-------------------

        r - Добавить прямоугольник

             b - Добавить ромб

           t - Добавить трапецию

             2 - Извлечь фигуру

           3 - Просмотреть очередь

             4 - Удалить очередь
```

```
                  x - выход

-------------------------------------------


r
Введите длину:
5
Введите ширину:
7
Добавлен прямоугольник

b
Введите сторону:
7
Введите высоту:
8
Добавлен ромб

t
Введите верх. основание.:
5
Введите нижн. основание:
10
Введите высоту:
4
Добавлена трапеция

3
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Трапеция
Верхнее основание: 5
Нижнее основание: 10
Высота: 4
Площадь: 30


2
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

3
Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Трапеция
Верхнее основание: 5
Нижнее основание: 10
Высота: 4
```

```
Площадь: 30


4
Очередь удалена

2
Очередь пуста

x
Очередь удалена
user@lubuntu:~/OOP/lab3$
```

Лабораторная работа №4

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Знакомство с шаблонами классов.
- Построение шаблонов динамических структур данных.

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ **шаблон класса-контейнера** первого уровня, содержащий **все три** фигуры класса фигуры, согласно вариантов задания (реализованную в ЛР1).

Классы должны удовлетворять следующим правилам:

- Требования к классам фигуры аналогичны требованиям из лабораторной работы 1.
- Шаблон класса-контейнера должен соджержать объекты используя std:shared_ptr<…>.
- Шаблон класса-контейнера должен иметь метод по добавлению фигуры в контейнер.
- Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера (опередеяется структурой контейнера).
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (опередеяется структурой контейнера).
- Шаблон класса-контейнера должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).
- Шаблон класса-контейнера должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Код программы:

TQueue.h

```
template <class T> class TQueue {
public:
    TQueue();

    void push(std::shared_ptr<T> &&figure);
    std::shared_ptr<T> pop();
    bool empty();
    template <class A> friend std::ostream& operator<<(std::ostream& os,const
TQueue<A>& queue);
    virtual ~TQueue();
private:

    std::shared_ptr<TQueueItem<T>> head;
    std::shared_ptr<TQueueItem<T>> tail;
};
```

TQueue.cpp

```
#ifndef TQUEUE_FUNCTIONS
#include "TQueue.h"

#else

template <class T> TQueue<T>::TQueue() {
    tail = nullptr;
    head = nullptr;
}

template <class T> std::ostream& operator<<(std::ostream& os, const
TQueue<T>& queue) {

    std::shared_ptr<TQueueItem<T>> item = queue.head;

    while(item != nullptr) {

        std::shared_ptr<Figure> t = item->GetFigure();
        t->Print();
        std::cout << '\n';
        item = item->GetNext();
    }

    return os;
}

template <class T> void TQueue<T>::push(std::shared_ptr<T> &&figure) {

    std::shared_ptr<TQueueItem<T>> other(new TQueueItem<T>(figure));
    if (empty()) {
        head = other;
        tail = other;
        std::shared_ptr<TQueueItem<T>> empty = nullptr;
        other->SetNext(empty);
        return;
    }
```

```
    tail->SetNext(other);
    tail = other;
    std::shared_ptr<TQueueItem<T>> empty = nullptr;
    tail->SetNext(empty);
}

template <class T> bool TQueue<T>::empty() {
    return head == nullptr;
}

template <class T> std::shared_ptr<T> TQueue<T>::pop() {

    std::shared_ptr<T> result;
    if (!empty()) {
        std::shared_ptr<TQueueItem<T>> old_head = head;
        head = head->GetNext();
        result = old_head->GetFigure();
        std::shared_ptr<TQueueItem<T>> empty = nullptr;
        old_head->SetNext(empty);
        return result;
    }
    else {
        std::cout << "Очередь пуста " << std::endl;
        return nullptr;
    }
}

template <class T> TQueue<T>::~TQueue() {
    tail = nullptr;
    head = nullptr;
    std::cout << "Очередь удалена" << std::endl;
}

#endif
```

TQueueitem.h

```
template <class T> class TQueueItem {
public:
    TQueueItem(const std::shared_ptr<T>& figure);
    template <class A> friend std::ostream& operator<<(std::ostream& os,
const TQueueItem<A>& obj);

    void SetNext(std::shared_ptr<TQueueItem> &next);
    std::shared_ptr<TQueueItem<T>> GetNext();
    std::shared_ptr<T> GetFigure() const;

    virtual ~TQueueItem();
private:
    std::shared_ptr<T> figure;
    std::shared_ptr<TQueueItem<T>> next;
};
```

TQueueitem.cpp

```
#ifndef TQUEUEITEM_FUNCTION
#include "TQueueItem.h"
#include <iostream>
```

```
#else

template <class T> TQueueItem<T>::TQueueItem(const std::shared_ptr<T>&
figure) {
    this->figure = figure;
    this->next = nullptr;
}

template <class T> void
TQueueItem<T>::SetNext(std::shared_ptr<TQueueItem<T>>& next) {
    this->next = next;
    return;
}

template <class T> std::shared_ptr<T> TQueueItem<T>::GetFigure() const {
    return this->figure;
}

template <class T> std::shared_ptr<TQueueItem<T>> TQueueItem<T>::GetNext() {
    return this->next;
}

template <class A> std::ostream& operator<<(std::ostream& os, const
TQueueItem<A>& obj) {
    os << *obj.figure << "\n";
    return os;
}

template <class T> TQueueItem<T>::~TQueueItem() {
}

#endif
```

Тестирование:

```
user@lubuntu:~/OOP/lab4$ ./main.exe
----------------------MENU-------------------

        r - Добавить прямоугольник

             b - Добавить ромб

           t - Добавить трапецию

             2 - Извлечь фигуру

           3 - Просмотреть очередь

             4 - Удалить очередь

                  x - выход

-----------------------------------------


r
Введите длину:
5
Введите ширину:
7
```

```
Добавлен прямоугольник

b
Введите сторону:
7
Введите высоту:
8
Добавлен ромб

t
Введите верх. основание.:
5
Введите нижн. основание:
10
Введите высоту:
7
Добавлена трапеция

3
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Трапеция
Верхнее основание: 5
Нижнее основание: 10
Высота: 7
Площадь: 52.5

2
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

3
Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Трапеция
Верхнее основание: 5
Нижнее основание: 10
Высота: 7
Площадь: 52.5


4
Очередь удалена

2
Очередь пуста

x
Очередь удалена
user@lubuntu:~/OOP/lab4$
```

Лабораторная работа №5

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Закрепление навыков работы с шаблонами классов.
- Построение итераторов для динамических структур данных.

ЗАДАНИЕ

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР№4) спроектировать и разработать Итератор для динамической структуры данных.

Итератор должен быть разработан в виде шаблона и должен уметь работать со всеми типами фигур, согласно варианту задания.

Итератор должен позволять использовать структуру данных в операторах типа for. Например:

```
for(auto i : stack)  std::cout << *i << std::endl;
```

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Код программы:

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H
#include <memory>
#include <iostream>

template <class node, class T> class TIterator {
public:

    TIterator(std::shared_ptr<node> n) {
        node_ptr = n;
    }

    std::shared_ptr<T> operator * () {
        return node_ptr->GetFigure();
    }

    std::shared_ptr<T> operator -> () {
        return node_ptr->GetFigure();
    }

    void operator ++ () {
        node_ptr = node_ptr->GetNext();
    }

    TIterator operator ++ (int) {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }

    bool operator == (TIterator const& i) {
        return node_ptr == i.node_ptr;
    }

    bool operator != (TIterator const& i) {
        return !(node_ptr == i.node_ptr);
    }

private:

    std::shared_ptr<node> node_ptr;
};

#endif
```

TQueue.h

```
TIterator<TQueueItem<T>,T> begin();
TIterator<TQueueItem<T>,T> end();
```

TQueue.cpp

```
template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::begin() {
    return TIterator<TQueueItem<T>,T>(head);
```

```
}

template <class T> TIterator<TQueueItem<T>,T> TQueue<T>::end() {
    return TIterator<TQueueItem<T>,T>(nullptr);
}
```

main.cpp

```
for (auto i: queue) {
     i->Print();
     std::cout << '\n';
}
```

Тестирование:

```
user@lubuntu:~/OOP/lab5$ ./main.exe
--------------------MENU------------------

        r - Добавить прямоугольник

             b - Добавить ромб

           t - Добавить трапецию

             2 - Извлечь фигуру

           3 - Просмотреть очередь

             4 - Удалить очередь

                  x - выход

------------------------------------------


r
Введите длину:
5
Введите ширину:
7
Добавлен прямоугольник

b
Введите сторону:
7
Введите высоту:
8
Добавлен ромб

t
Введите верх. основание.:
5
Введите нижн. основание:
10
Введите высоту:
5
Добавлена трапеция

3
Прямоугольник
```

```
Длина: 5
Высота: 7
Площадь: 35

Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Трапеция
Верхнее основание: 5
Нижнее основание: 10
Высота: 5
Площадь: 37.5


2
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

3
Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Трапеция
Верхнее основание: 5
Нижнее основание: 10
Высота: 5
Площадь: 37.5


4
Очередь удалена

2
Очередь пуста

x
Очередь удалена
user@lubuntu:~/OOP/lab5$
```

Лабораторная работа №6

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Закрепление навыков по работе с памятью в C++.
- Создание аллокаторов памяти для динамических структур данных.

ЗАДАНИЕ

Используя структуры данных, разработанные для предыдущей лабораторной работы (ЛР№5) спроектировать и разработать аллокатор памяти для динамической структуры данных.

Цель построения аллокатора – минимизация вызова операции **malloc**. Аллокатор должен выделять большие блоки памяти для хранения фигур и при создании новых фигур-объектов выделять место под объекты в этой памяти.

Алокатор должен хранить списки использованных/свободных блоков. Для хранения списка свободных блоков нужно применять динамическую структуру данных (контейнер 2-го уровня, согласно варианта задания).

Для вызова аллокатора должны быть переопределены оператор **new** и **delete** у классов-фигур.

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Код программы:

Block.h

```
#ifndef BLOCK_H
#define BLOCK_H
#include <cstdlib>
#include "TreeNode.h"

class TAllocationBlock {
public:
    TAllocationBlock(size_t _size, size_t _count);
    void *allocate();
    void deallocate(void *pointer);
    bool has_free_blocks();

    Tree* Insert(Tree* tree, size_t key);
    Tree* Search(Tree* tree, size_t key);
    Tree* CreateTree(size_t left_border, size_t right_border, Tree* root);
    void DeleteTree(Tree* tree);

    virtual ~TAllocationBlock();

private:
    size_t size;
    size_t count;

    char *used_block;
    Tree *root;
    int first;
    int last;

    size_t count_free;
};

#endif
```

Block.cpp

```
#include "Block.h"
#include <iostream>
#include <math.h>

TAllocationBlock::TAllocationBlock(size_t _size, size_t _count):size(_size),
count(_count) {
    used_block = (char*)malloc(size * count);
    root = nullptr;
    root = CreateTree(1,count,root);
    count_free = count;
    first = 0;
    last = 1;
}

void *TAllocationBlock::allocate() {
    void *result = nullptr;
    if (count_free > 0) {
        if (count_free == count) {
            Tree* tree = Search(root,1);
            result = tree->free_block;
```

```
            --count_free;
            first = 1;
            last = 2;
            std::cout << "Память выделена" << std::endl;
        }
        else {
            Tree* tree = Search(root,last);
            result = tree->free_block;
            --count_free;
            last = (last % count) + 1;
             std::cout << "Память выделена" << std::endl;
        }
    }
    else {
        std::cout << "Памяти больше нет" << std::endl;
        throw std::bad_alloc();
    }

    return result;
}

void TAllocationBlock::deallocate(void* pointer) {
    Tree* tree = Search(root, first);
    tree->free_block = pointer;
    ++count_free;
    first = (first % count) + 1;
     std::cout << "Память освобождена" << std::endl;
}

bool TAllocationBlock::has_free_blocks() {
    return count_free > 0;
}

TAllocationBlock::~TAllocationBlock() {
    DeleteTree(root);
    free(used_block);
}

Tree* TAllocationBlock::Insert(Tree* tree, size_t key) {
    if(tree == nullptr) {
        Tree* newTree = (Tree*)malloc(sizeof(Tree));
        newTree->key = key;
        newTree->left = nullptr;
        newTree->right = nullptr;
        newTree->free_block = used_block + (key - 1) * size;
        return newTree;
    }
    if(tree->key == key) {
        return tree;
    }
    if(key < tree->key) {
        tree->left = Insert(tree->left, key);
    }else {
        tree->right = Insert(tree->right, key);
    }
    return tree;
}

Tree* TAllocationBlock::CreateTree(size_t left_border, size_t right_border,
Tree* tree) {
    if (left_border == right_border) {
        tree = Insert(tree,left_border);
        return tree;
```

```
    }
    int average = 0;
    int con = 0;
    for (int i = left_border; i <= right_border; ++i) {
        average += i;
        ++con;
    }
    if (con == 0 ) {
        return tree;
    }
    average = ceil(average / con);
    tree = Insert(tree,average);
    CreateTree(left_border, average - 1, tree);
    CreateTree(average + 1, right_border, tree);
    return tree;
}

Tree* TAllocationBlock::Search(Tree* tree, size_t key) {
    if(tree == nullptr) {
        return nullptr;
    }
    if(tree->key == key) {
        return tree;
    }
    if(key < tree->key) {
        Search(tree->left, key);
    }else {
        Search(tree->right, key);
    }
}

void TAllocationBlock::DeleteTree(Tree* tree) {

    if (tree != nullptr) {
        DeleteTree(tree->left);
        DeleteTree(tree->right);
        free(tree);
    }
}
```

TQueueitem.h

```
static TAllocationBlock queueItem_allocator;
```

TQueueitem.cpp

```
template <class T> TAllocationBlock
TQueueItem<T>::queueItem_allocator(sizeof(TQueueItem<T>),100);

template <class T> void * TQueueItem<T>::operator new(size_t size) {
    return queueItem_allocator.allocate();
}

template <class T> void TQueueItem<T>::operator delete(void *p) {
    queueItem_allocator.deallocate(p);
}
```

Тестирование:

```
user@lubuntu:~/OOP/lab6$ ./main.exe
--------------------MENU------------------

        r - Добавить прямоугольник

             b - Добавить ромб

           t - Добавить трапецию

             2 - Извлечь фигуру

           3 - Просмотреть очередь

             4 - Удалить очередь

                  x - выход

------------------------------------------


r
Введите длину:
5
Введите ширину:
7
Память выделена
Добавлен прямоугольник

b
Введите сторону:
7
Введите высоту:
8
Память выделена
Добавлен ромб

t
Введите верх. основание.:
7
Введите нижн. основание:
10
Введите высоту:
8
Память выделена
Добавлена трапеция

3
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Трапеция
Верхнее основание: 7
Нижнее основание: 10
```

```
Высота: 8
Площадь: 68


2
Память освобождена
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

3
Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Трапеция
Верхнее основание: 7
Нижнее основание: 10
Высота: 8
Площадь: 68


4
Память освобождена
Память освобождена
Очередь удалена

2
Очередь пуста

x
Очередь удалена
user@lubuntu:~/OOP/lab6$
```

Лабораторная работа №7

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Создание сложных динамических структур данных.
- Закрепление принципа OCP.

ЗАДАНИЕ

Необходимо реализовать динамическую структуру данных – «Хранилище объектов» и алгоритм работы с ней. «Хранилище объектов» представляет собой контейнер, одного из следующих видов (Контейнер 1-го уровня):

1. Массив

2. Связанный список

3. Бинарное- Дерево.

4. N-Дерево (с ограничением не больше 4 элементов на одном уровне).

5. Очередь

6. Стек

Каждым элементом контейнера, в свою, является динамической структурой данных одного из следующих видов (Контейнер 2-го уровня):

1. Массив

2. Связанный список

3. Бинарное- Дерево

4. N-Дерево (с ограничением не больше 4 элементов на одном уровне).

5. Очередь

6. Стек

Таким образом у нас получается контейнер в контейнере. Т.е. для варианта (1,2) это будет массив, каждый из элементов которого – связанный список. А для варианта (5,3) – это очередь из бинарных деревьев.

Элементом второго контейнера является объект-фигура, определенная вариантом задания.

При этом должно выполняться правило, что количество объектов в контейнере второго уровня не больше **5**. Т.е. если нужно хранить больше 5 объектов, то создается еще один контейнер второго уровня. Например, для варианта (1,2) добавление объектов будет выглядеть следующим образом:

1. Вначале массив пустой.

2. Добавляем Объект1: В массиве по индексу 0 создается элемент с типом список, в список

добавляется Объект 1.

3. Добавляем Объект2: Объект добавляется в список, находящийся в массиве по индекс 0.

4. Добавляем Объект3: Объект добавляется в список, находящийся в массиве по индекс 0.

5. Добавляем Объект4: Объект добавляется в список, находящийся в массиве по индекс 0.
6. Добавляем Объект5: Объект добавляется в список, находящийся в массиве по индекс 0.

7. Добавляем Объект6: В массиве по индексу 1 создается элемент с типом список, в список добавляется Объект 6.

Объекты в контейнерах второго уровня должны быть отсортированы по возрастанию площади объекта (в том числе и для деревьев).

При удалении объектов должно выполняться правило, что контейнер второго уровня не должен быть пустым. Т.е. если он становится пустым, то он должен удалится.

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера (1-го и 2-го уровня).
- Удалять фигуры из контейнера по критериям:
    - По типу (например, все квадраты).
    - По площади (например, все объекты с площадью меньше чем заданная).

Код программы:

IRemoveCriteria.h

```
#ifndef IREMOVECRITERIA_H
#define IREMOVECRITERIA_H
#include <memory>

template <class TT> class IRemoveCriteria {
public:
    virtual bool isIt(std::shared_ptr<TT> obj) {};
};

#endif
```

IRemoveCriteriaByType

```
#ifndef IREMOVECRITERIABYTYPE_H
#define IREMOVECRITERIABYTYPE_H

#include "IRemoveCriteria.h"

template <class TT> class IRemoveCriteriaAll : public IRemoveCriteria<TT> {
public:
    IRemoveCriteriaAll(int _type) : type(_type) {};
    bool isIt(std::shared_ptr<TT> obj) override {
        return type == obj->GetType();
    }
private:
    int type;
};

#endif
```

IRemoveCriteriaBySquare

```
#ifndef IREMOVECRITERIABYSQUARE_H
#define IREMOVECRITERIABYSQUARE_H

#include "IRemoveCriteria.h"

template <class TT> class IRemoveCriteriaBySquare : public
IRemoveCriteria<TT> {
public:
    IRemoveCriteriaBySquare(double _square, char _symbol) : square(_square),
symbol(_symbol) {};
    bool isIt(std::shared_ptr<TT> obj) override {
        switch(symbol) {
            case '<' : {
                return obj->Square() < square;
                break;
            }
            case '=' : {
                return obj->Square() == square;
                break;
            }
            case '>' : {
```

```
                return obj->Square() > square;
                break;
            }
        }
    }
private:
    double square;
    char symbol;
};

#endif
```

TBinaryTree.h

```
#ifndef TBINARYTREE_H
#define TBINARYTREE_H
#include "TreeNode.h"
#include <iostream>
#include "IRemoveCriteriaAll.h"
#include "IRemoveCriteriaBySquare.h"

template <class TT> class TBinaryTree {
public:
    TBinaryTree(std::shared_ptr<TT> _figure);
    void Insert(std::shared_ptr<TT> figure);
    void PrintTree();
    int GetSize();
    std::shared_ptr<TreeNode<TT>> GetRoot();
    void Inorder(IRemoveCriteria<TT> * criteria, std::shared_ptr<TT> temp[],
int &i);
    bool Empty();


    ~TBinaryTree();
private:
    void Insert(std::shared_ptr<TreeNode<TT>> node, std::shared_ptr<TT>
figure);
    void PrintTree(std::shared_ptr<TreeNode<TT>> node, int tab);
    void Inorder(std::shared_ptr<TreeNode<TT>> node, IRemoveCriteria<TT> *
criteria, std::shared_ptr<TT> temp[], int &i);

    std::shared_ptr<TreeNode<TT>> root;
    int size;

};


#define TREE_FUNCTIONS
#include "TBinaryTree.cpp"

#endif
```

TBinaryTree.cpp

```
#ifndef TREE_FUNCTIONS
#include "TBinaryTree.h"

#else
```

```
template <class TT> TBinaryTree<TT>::TBinaryTree(std::shared_ptr<TT> _figure)
{
    root.reset(new TreeNode<TT>(_figure));
    size = 1;
}

template <class TT> void TBinaryTree<TT>::Insert(std::shared_ptr<TT> figure)
{
    Insert(root, figure);
    return;
}

template <class TT> void
TBinaryTree<TT>::Insert(std::shared_ptr<TreeNode<TT>> node,
std::shared_ptr<TT> figure) {

    if (node->figure->Square() < figure->Square() && node->right == nullptr)
{
        node->right.reset(new TreeNode<TT>(figure));
        ++size;
        return;
    }
    if (node->figure->Square() >= figure->Square() && node->left == nullptr)
{
        node->left.reset(new TreeNode<TT>(figure));
        ++size;
        return;
    }
    if (node->figure->Square() < figure->Square()) {
        Insert(node->right, figure);
    }
    if (node->figure->Square() >= figure->Square()) {
        Insert(node->left, figure);
    }
    return;
}

template <class TT> int TBinaryTree<TT>::GetSize() {
    return size;
}

template <class TT> bool TBinaryTree<TT>::Empty() {
    return size == 0;
}

template <class TT> void TBinaryTree<TT>::PrintTree() {
    puts("---------------------------------------\n");
    PrintTree(root,0);
    return;
}

template <class TT> void
TBinaryTree<TT>::PrintTree(std::shared_ptr<TreeNode<TT>> node,int tab) {
    if (node == nullptr) {
        return;
    }
    PrintTree(node->left, tab + 0);
    for (int i = 0; i < tab; ++i){
        putchar(' ');
    }
    node->figure->Print();
    std::cout << std::endl;
```

```
    PrintTree(node->right, tab + 0);
}

template <class TT> void TBinaryTree<TT>::Inorder(IRemoveCriteria<TT> *
criteria, std::shared_ptr<TT> temp[], int &i) {
    Inorder(root, criteria, temp, i);
    return;
}

template <class TT> void
TBinaryTree<TT>::Inorder(std::shared_ptr<TreeNode<TT>> node,
IRemoveCriteria<TT> * criteria, std::shared_ptr<TT> temp[], int &i) {
    if (node == nullptr) {
        return;
    }
    Inorder(node->left, criteria, temp, i);
    if (!criteria->isIt(node->figure)) {
        temp[i] = node->figure;
        ++i;
    }
    Inorder(node->right, criteria, temp, i);
}

template <class TT> std::shared_ptr<TreeNode<TT>> TBinaryTree<TT>::GetRoot()
{
    return root;
}

template <class TT> TBinaryTree<TT>::~TBinaryTree() {

}

#endif
```

TreeNode.h

```
#ifndef TREENODE_H
#define TREENODE_H
#include <cstdlib>
#include <memory>

template <class TT> class TreeNode {
public:
    TreeNode(std::shared_ptr<TT> _figure) : figure(_figure), left(nullptr),
right(nullptr) {};

    std::shared_ptr<TreeNode<TT>> left;
    std::shared_ptr<TreeNode<TT>> right;
    std::shared_ptr<TT> figure;

};

#endif
```

TQueue.h

```
template <class T, class TT> class TQueue {
public:
```

```
    TQueue();

    void PushSubitem(std::shared_ptr<TT> figure);
    void DeleteSubitem(IRemoveCriteria<TT> * criteria);
    void Push(std::shared_ptr<TT> figure);
    void Pop();
    bool Empty();

    template <class A, class AA> friend std::ostream&
operator<<(std::ostream& os,const TQueue<A,AA>& queue);

    TIterator<TQueueItem<T,TT>,T> begin();
    TIterator<TQueueItem<T,TT>,T> end();

    std::shared_ptr<TQueueItem<T,TT>> head;
    std::shared_ptr<TQueueItem<T,TT>> tail;

    virtual ~TQueue();
private:


};
```

TQueue.cpp

```
#ifndef TQUEUE_FUNCTIONS
#include "TQueue.h"

#else

template <class T, class TT> TQueue<T, TT>::TQueue() {
    tail = nullptr;
    head = nullptr;
}

template <class A, class AA> std::ostream& operator<<(std::ostream& os, const
TQueue<A, AA>& queue) {
     for (auto i: queue) {
        i->PrintTree();
        std::cout << '\n';
    }
}

template <class T, class TT> void TQueue<T,TT>::Push(std::shared_ptr<TT>
figure) {
    std::shared_ptr<TQueueItem<T,TT>> other(new TQueueItem<T,TT>());
    std::shared_ptr<TBinaryTree<TT>> newTree(new TBinaryTree<TT>(figure));
    other->SetTree(newTree);
    if (Empty()) {
        head = other;
        tail = other;
        std::shared_ptr<TQueueItem<T,TT>> empty = nullptr;
        other->SetNext(empty);
        return;
    }

    tail->SetNext(other);
    tail = other;
    std::shared_ptr<TQueueItem<T,TT>> empty = nullptr;
    tail->SetNext(empty);
}
```

```
template <class T, class TT> void TQueue<T,TT>::Pop() {
    if (Empty()) {
        std::cout << "Очередь пуста" << std::endl;
        return;
    }
    else {
        std::shared_ptr<TQueueItem<T,TT>> old_head = head;
        head = head->GetNext();
        std::shared_ptr<TQueueItem<T,TT>> empty = nullptr;
        old_head->SetNext(empty);
        return;
    }
}

template <class T, class TT> void
TQueue<T,TT>::PushSubitem(std::shared_ptr<TT> figure) {
    if (Empty()) {
        Push(figure);
    }

    else if (tail->TGetSize() < 5) {
        tail->GetTree()->Insert(figure);
    }
    else {
        Push(figure);
    }
    return;
}

template <class T, class TT> void
TQueue<T,TT>::DeleteSubitem(IRemoveCriteria<TT> * criteria) {
        if (Empty()) {
            std::cout << "Очередь пуста" << std::endl;
            return;
        }

       int size = head->TGetSize();
       int count = 0;
       std::shared_ptr<TT> *temp = new std::shared_ptr<TT>[size];
       head->Inorder(criteria, temp, count);

        if (count == 0) {
            Pop();
            return;
        }

       std::shared_ptr<TBinaryTree<TT>> newTree(new
TBinaryTree<TT>(temp[0]));
       for (int j = 1; j < count; ++j) {
           newTree->Insert(temp[j]);
       }
       head->Delete();
       head->SetTree(newTree);

    return;
}

template <class T, class TT> bool TQueue<T,TT>::Empty() {
    return head == nullptr;
}
```

```
template <class T, class TT> TIterator<TQueueItem<T,TT>,T>
TQueue<T,TT>::begin() {
    return TIterator<TQueueItem<T,TT>,T>(head);
}

template <class T, class TT> TIterator<TQueueItem<T,TT>,T>
TQueue<T,TT>::end() {
    return TIterator<TQueueItem<T,TT>,T>(nullptr);
}

template <class T, class TT> TQueue<T,TT>::~TQueue() {
    tail = nullptr;
    head = nullptr;
    std::cout << "Очередь удалена" << std::endl;
}

#endif
```

TQueueitem.h

```
#ifndef TQUEUEITEM_H
#define TQUEUEITEM_H

#include "Rectangle.h"
#include "Rhomb.h"
#include "Trapeze.h"
#include "TBinaryTree.h"
#include <memory>
#include <cstdlib>

template <class T, class TT> class TQueueItem {
public:
    TQueueItem();

    void SetNext(std::shared_ptr<TQueueItem<T,TT>>& next);
    std::shared_ptr<TQueueItem<T,TT>> GetNext();
    std::shared_ptr<TBinaryTree<TT>> GetTree();
    void SetTree(std::shared_ptr<TBinaryTree<TT>> newTree);
    void Inorder(IRemoveCriteria<TT> * criteria, std::shared_ptr<TT> temp[],
int &i);
    int TGetSize();
    void TPrintTree();
    void Delete();

    template <class A, class AA> friend std::ostream&
operator<<(std::ostream& os, const TQueueItem<A,AA>& obj);

    virtual ~TQueueItem();
private:
    std::shared_ptr<TQueueItem<T,TT>> next;
    std::shared_ptr<TBinaryTree<TT>> tree;

};


#define TQUEUEITEM_FUNCTION
#include "TQueueItem.cpp"
#endif
```

TQueueitem.cpp

```
#ifndef TQUEUEITEM_FUNCTION
#include "TQueueItem.h"
#include <iostream>

#else

template <class T, class TT> TQueueItem<T,TT>::TQueueItem() {
    this->next = nullptr;
}

template <class T, class TT> void
TQueueItem<T,TT>::SetNext(std::shared_ptr<TQueueItem<T,TT>>& next) {
    this->next = next;
    return;
}

template <class T,class TT> std::shared_ptr<TQueueItem<T,TT>>
TQueueItem<T,TT>::GetNext() {
    return this->next;
}

template <class T,class TT> void
TQueueItem<T,TT>::SetTree(std::shared_ptr<TBinaryTree<TT>> newTree) {
    this->tree = newTree;
}

template <class T, class TT> std::shared_ptr<TBinaryTree<TT>>
TQueueItem<T,TT>::GetTree() {
    return this->tree;
}

template <class T, class TT> int TQueueItem<T,TT>::TGetSize() {
    return tree->GetSize();
}

template <class T, class TT> void
TQueueItem<T,TT>::Inorder(IRemoveCriteria<TT> * criteria, std::shared_ptr<TT>
temp[], int &i) {
    tree->Inorder(criteria, temp, i);
}

template <class T, class TT> void TQueueItem<T,TT>::TPrintTree() {
    if (tree->GetSize() == 0) {
        return;
    }
    tree->PrintTree();
}

template <class T, class TT> void TQueueItem<T,TT>::Delete() {
    tree->~TBinaryTree();
}

template <class A, class AA> std::ostream& operator<<(std::ostream& os, const
TQueueItem<A,AA>& obj) {
    os << obj->PrintTree() << std::endl;
    return os;
}

template <class T, class TT> TQueueItem<T,TT>::~TQueueItem() {
}
```

```
#endif
```

Rectangle.h

```
int GetType() override;
```

Rectangle.cpp

```
int Rectangle::GetType() {
    return 1;
}
```

Trapeze.h

```
int GetType() override;
```

Trapeze.h

```
int Rhomb::GetType() {
    return 2;
}
```

Rhomb.h

```
int GetType() override;
```

Rhomb.cpp

```
int Rhomb::GetType() {
    return 3;
}
```

Titerator.h

```
template <class node, class T> class TIterator {
public:

    TIterator(std::shared_ptr<node> n) {
        node_ptr = n;
    }

    std::shared_ptr<node> operator * () {
       return node_ptr;
    }

    std::shared_ptr<node> operator -> () {
    return node_ptr;
    }

    void operator ++ () {
        node_ptr = node_ptr->GetNext();
    }

    TIterator operator ++ (int) {
        TIterator iter(*this);
        ++(*this);
        return iter;
    }
```

```
    bool operator == (TIterator const& i) {
        return node_ptr == i.node_ptr;
    }

    bool operator != (TIterator const& i) {
        return !(node_ptr == i.node_ptr);
    }

private:

    std::shared_ptr<node> node_ptr;
}
```

main.cpp

```
#include <cstdlib>
#include <iostream>
#include "TQueueItem.h"
#include "TQueue.h"
#include <memory>
#include "IRemoveCriteriaAll.h"
#include "IRemoveCriteriaBySquare.h"


int main(int argc, char** argv) {

    char choice;
    TQueue<TreeNode<Figure>,Figure> queue;

    puts("----------------------MENU-------------------\n");
    puts("        r - Добавить прямоугольник          \n");
    puts("             b - Добавить ромб              \n");
    puts("           t - Добавить трапецию            \n");
    puts("             2 - Извлечь голову             \n");
    puts("           3 - Просмотреть очередь          \n");
    puts("             4 - Удалить очередь            \n");
    puts("       5 - Удаление элементов по площади    \n");
    puts("         6 - Удаление элементов по типу     \n");
    puts("                   x - выход                \n");
    puts("-----------------------------------------\n");

    do {
        std::cout << '\n';
        std::cin >> choice;

        switch(choice) {
            case 'r': {
                queue.PushSubitem(std::shared_ptr<Rectangle>(new
Rectangle(std::cin)));
                std::cout << "Добавлен прямоугольник " << std::endl;
                break;
            }
            case 'b': {
                queue.PushSubitem(std::shared_ptr<Rhomb>(new
Rhomb(std::cin)));
                std::cout << "Добавлен ромб " << std::endl;
                break;
            }
            case 't': {
                queue.PushSubitem(std::shared_ptr<Trapeze>(new
Trapeze(std::cin)));
                std::cout << "Добавлена трапеция " << std::endl;
                break;
```

```
            }
            case '2': {
                queue.Pop();
                break;
            }
            case '3': {
                for (auto i: queue) {
                    i->TPrintTree();
                    std::cout << '\n';
                }
                std::cout << '\n';
                break;
            }
            case '4': {
                queue.~TQueue();
                break;
            }
            case '5': {
                char symbol;
                double square;
                std::cout << "Введите площадь и символ сравнения\n";
                std::cin >> square >> symbol;
                IRemoveCriteriaBySquare<Figure> criteria(square, symbol);
                queue.DeleteSubitem(&criteria);
                break;
            }
            case '6': {
                int type;
                std::cout << "Введите тип фигуры: 1 - Прямоугольник, 2 -
Трапеция, 3 - Ромб\n";
                std::cin >> type;
                IRemoveCriteriaAll<Figure> criteria(type);
                queue.DeleteSubitem(&criteria);
                break;
            }
            case 'x':{
                break;
            }
            default:{
                std::cout << "Нет варианта" << std::endl;
                break;
            }
        }
    } while (choice != 'x');

    return 0;
}
```

Тестирование:

```
user@lubuntu:~/OOP/lab7$ ./main.exe
----------------------MENU-------------------

        r - Добавить прямоугольник

             b - Добавить ромб

           t - Добавить трапецию

             2 - Извлечь голову
```

```
            3 - Просмотреть очередь

              4 - Удалить очередь

        5 - Удаление элементов по площади

          6 - Удаление элементов по типу

                   x - выход

-------------------------------------------


r
Введите длину:
5
Введите ширину:
7
Добавлен прямоугольник

b
Введите сторону:
7
Введите высоту:
8
Добавлен ромб

t
Введите верх. основание.:
5
Введите нижн. основание:
12
Введите высоту:
5
Добавлена трапеция

r
Введите длину:
8
Введите ширину:
9
Добавлен прямоугольник

b
Введите сторону:
4
Введите высоту:
12
Добавлен ромб

t
Введите верх. основание.:
12
Введите нижн. основание:
20
Введите высоту:
5
Добавлена трапеция

r
Введите длину:
7
Введите ширину:
```

```
5
Добавлен прямоугольник

b
Введите сторону:
9
Введите высоту:
12
Добавлен ромб

r
Введите длину:
15
Введите ширину:
5
Добавлен прямоугольник

3
----------------------------------------

Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

Трапеция
Верхнее основание: 5
Нижнее основание: 12
Высота: 5
Площадь: 42.5

Ромб
Сторона ромба: 4
Высота: 12
Площадь: 48

Ромб
Сторона ромба: 7
Высота: 8
Площадь: 56

Прямоугольник
Длина: 8
Высота: 9
Площадь: 72


----------------------------------------

Прямоугольник
Длина: 7
Высота: 5
Площадь: 35

Прямоугольник
Длина: 15
Высота: 5
Площадь: 75

Трапеция
Верхнее основание: 12
Нижнее основание: 20
Высота: 5
```

```
Площадь: 80

Ромб
Сторона ромба: 9
Высота: 12
Площадь: 108



2

3
---------------------------------------

Прямоугольник
Длина: 7
Высота: 5
Площадь: 35

Прямоугольник
Длина: 15
Высота: 5
Площадь: 75

Трапеция
Верхнее основание: 12
Нижнее основание: 20
Высота: 5
Площадь: 80

Ромб
Сторона ромба: 9
Высота: 12
Площадь: 108



5
Введите площадь и символ сравнения
80 <

3
---------------------------------------

Трапеция
Верхнее основание: 12
Нижнее основание: 20
Высота: 5
Площадь: 80

Ромб
Сторона ромба: 9
Высота: 12
Площадь: 108



b
Введите сторону:
4
```

```
Введите высоту:
15
Добавлен ромб

3
----------------------------------------

Ромб
Сторона ромба: 4
Высота: 15
Площадь: 60

Трапеция
Верхнее основание: 12
Нижнее основание: 20
Высота: 5
Площадь: 80

Ромб
Сторона ромба: 9
Высота: 12
Площадь: 108




6
Введите тип фигуры: 1 - Прямоугольник, 2 - Трапеция, 3 - Ромб
3

3
----------------------------------------

Трапеция
Верхнее основание: 12
Нижнее основание: 20
Высота: 5
Площадь: 80




4
Очередь удалена

2
Очередь пуста

x
Очередь удалена
user@lubuntu:~/OOP/lab7$
```

Лабораторная работа №8

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Знакомство с параллельным программированием в C++.

ЗАДАНИЕ

Используя структуры данных, разработанные для лабораторной работы №6 (контейнер первого уровня и классы-фигуры) разработать алгоритм быстрой сортировки для класса-контейнера .

Необходимо разработать два вида алгоритма:

- Обычный, без параллельных вызовов.
- С использованием параллельных вызовов. В этом случае, каждый рекурсивный вызов сортировки должен создаваться в отдельном потоке.

Для создания потоков использовать механизмы:

- future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.
- Проводить сортировку контейнера

Код программы:

Rectangle.h

```
bool operator==(const Figure& right);
bool operator<(const Figure& right);
bool operator>(const Figure& right);
bool operator<=(const Figure& right);
bool operator>=(const Figure& right);
operator double () const;
```

Rectangle.cpp

```
    return length * height;
}

bool Rectangle::operator==(const Figure& right) {
    return (double)(*this) == (double)(right);
}

bool Rectangle::operator<(const Figure& right) {
    return (double)(*this) < (double)(right);
}

bool Rectangle::operator>(const Figure& right) {
    return (double)(*this) > (double)(right);
}

bool Rectangle::operator<=(const Figure& right) {
    return (double)(*this) <= (double)(right);
}

bool Rectangle::operator>=(const Figure& right) {
    return (double)(*this) >= (double)(right);
}
```

Trapeze.h

```
bool operator==(const Figure& right);
bool operator<(const Figure& right);
bool operator>(const Figure& right);
bool operator<=(const Figure& right);
bool operator>=(const Figure& right);
```

Trapeze.cpp

```
Trapeze::operator double () const {
    return (up_base + low_base) * 0.5 * height;
}

bool Trapeze::operator==(const Figure& right) {
    return (double)(*this) == (double)(right);
}

bool Trapeze::operator<(const Figure& right) {
    return (double)(*this) < (double)(right);
```

```
}

bool Trapeze::operator>(const Figure& right) {
    return (double)(*this) > (double)(right);
}

bool Trapeze::operator<=(const Figure& right) {
    return (double)(*this) <= (double)(right);
}

bool Trapeze::operator>=(const Figure& right) {
    return (double)(*this) >= (double)(right);
}
```

Rhomb.h

```
bool operator==(const Figure& right);
bool operator<(const Figure& right);
bool operator>(const Figure& right);
bool operator<=(const Figure& right);
bool operator>=(const Figure& right);
operator double () const;
```

Rhomb.cpp

```
Rhomb::operator double () const {
    return rhomb_side * rhomb_height;
}

bool Rhomb::operator==(const Figure& right) {
    return (double)(*this) == (double)(right);
}

bool Rhomb::operator<(const Figure& right) {
    return (double)(*this) < (double)(right);
}

bool Rhomb::operator>(const Figure& right) {
    return (double)(*this) > (double)(right);
}

bool Rhomb::operator<=(const Figure& right) {
    return (double)(*this) <= (double)(right);
}

bool Rhomb::operator>=(const Figure& right) {
    return (double)(*this) >= (double)(right);
}
```

TQueue.h

```
template <class T> class TQueue {
public:
    TQueue();

    void push(std::shared_ptr<T> &&figure);
    std::shared_ptr<T> pop();
```

```
    bool empty();
    size_t size();
    void sort();
    void sort_parallel();
    template <class A> friend std::ostream& operator<<(std::ostream& os,const
TQueue<A>& queue);
    std::shared_ptr<T> operator[] (size_t ind);

    TIterator<TQueueItem<T>,T> begin();
    TIterator<TQueueItem<T>,T> end();

    virtual ~TQueue();
private:

    std::shared_ptr<TQueueItem<T>> head;
    std::shared_ptr<TQueueItem<T>> tail;
    std::future<void> sort_in_background();
};
```

TQueue.cpp

```
template <class T> std::shared_ptr<T> TQueue<T>::operator[] (size_t ind) {
    if (ind > size() - 1) throw std::invalid_argument("Индекс больше размера
очереди");
    size_t i = 0;
    for (auto a: *this) {
        if (i == ind) return a;
        ++i;
    }
    return std::shared_ptr<T>(nullptr);
}

template <class T> size_t TQueue<T>::size() {
    int result = 0;
    for (auto a: *this) ++result;
    return result;
}

template <class T> void TQueue<T>::sort() {
    if (size() > 1) {
        std::shared_ptr<T> middle = pop();
        TQueue<T> left, right;

        while (!empty()) {
            std::shared_ptr<T> item = pop();
            if (*item < *middle) {
                left.push(std::move(item));
            }
            else {
                right.push(std::move(item));
            }
        }
        left.sort();
        right.sort();

        while (!left.empty()) push(std::move(left.pop()));
        push(std::move(middle));
        while (!right.empty()) push(std::move(right.pop()));
    }
}
```

```
template <class T> std::future<void> TQueue<T>::sort_in_background() {
    std::packaged_task<void(void)>
    task(std::bind(std::mem_fn(&TQueue<T>::sort_parallel), this));
    std::future<void> res(task.get_future());
    std::thread th(std::move(task));
    th.detach();
    return res;
}

template <class T> void TQueue<T>::sort_parallel() {
    if (size() > 1) {
        std::shared_ptr<T> middle = pop();
        TQueue<T> left, right;

        while(!empty()) {
            std::shared_ptr<T> item = pop();
            if (*item < *middle) {
                left.push(std::move(item));
            }
            else {
                right.push(std::move(item));
            }
        }
        std::future<void> left_res = left.sort_in_background();
        std::future<void> right_res = right.sort_in_background();

        left_res.get();
        while (!left.empty()) push(std::move(left.pop()));
        push(std::move(middle));
        right_res.get();
        while (!right.empty()) push(std::move(right.pop()));
    }
}
```

main.cpp

```
case 's':{
    queue.sort();
    break;
}
case 'p':{
    queue.sort_parallel();
    break;
}
```

Тестирование:

```
user@lubuntu:~/OOP/lab8$ ./main.exe
----------------------MENU-------------------

        r - Добавить прямоугольник

             b - Добавить ромб

           t - Добавить трапецию

             2 - Извлечь фигуру
```

```
            3 - Просмотреть очередь

              4 - Удалить очередь

                s - Сортировка

          p - Параллельная сортировка

                    x - выход

-------------------------------------------

r
Введите длину:
5
Введите ширину:
7
Добавлен прямоугольник

b
Введите сторону:
12
Введите высоту:
15
Добавлен ромб

r
Введите длину:
20
Введите ширину:
30
Добавлен прямоугольник

5
Нет варианта

t
Введите верх. основание.:
12
Введите нижн. основание:
20
Введите высоту:
4
Добавлена трапеция

3
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

Ромб
Сторона ромба: 12
Высота: 15
Площадь: 180

Прямоугольник
Длина: 20
Высота: 30
Площадь: 600

Трапеция
Верхнее основание: 12
```

```
Нижнее основание: 20
Высота: 4
Площадь: 64


s

3
Прямоугольник
Длина: 5
Высота: 7
Площадь: 35

Трапеция
Верхнее основание: 12
Нижнее основание: 20
Высота: 4
Площадь: 64

Ромб
Сторона ромба: 12
Высота: 15
Площадь: 180

Прямоугольник
Длина: 20
Высота: 30
Площадь: 600
4
r
Введите длину:
19
Введите ширину:
8
Добавлен прямоугольник

b
Введите сторону:
15
Введите высоту:
6
Добавлен ромб

t
Введите верх. основание.:
15
Введите нижн. основание:
40
Введите высоту:
5
Добавлена трапеция

r
Введите длину:
4
Введите ширину:
5
Добавлен прямоугольник

b
Введите сторону:
17
Введите высоту:
```

```
5
Добавлен ромб

3
Прямоугольник
Длина: 19
Высота: 8
Площадь: 152

Ромб
Сторона ромба: 15
Высота: 6
Площадь: 90

Трапеция
Верхнее основание: 15
Нижнее основание: 40
Высота: 5
Площадь: 137.5

Прямоугольник
Длина: 4
Высота: 5
Площадь: 20

Ромб
Сторона ромба: 17
Высота: 5
Площадь: 85

p

3
Прямоугольник
Длина: 4
Высота: 5
Площадь: 20

Ромб
Сторона ромба: 17
Высота: 5
Площадь: 85

Ромб
Сторона ромба: 15
Высота: 6
Площадь: 90

Трапеция
Верхнее основание: 15
Нижнее основание: 40
Высота: 5
Площадь: 137.5

Прямоугольник
Длина: 19
Высота: 8
Площадь: 152

4

2
Очередь пуста
```

Лабораторная работа №9

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

- Знакомство с лямбда-выражениями

ЗАДАНИЕ

Используя структуры данных, разработанные для лабораторной работы №6 (контейнер первого уровня и классы-фигуры) необходимо разработать:

- Контейнер второго уровня с использованием шаблонов.
- Реализовать с помощью лямбда-выражений набор команд, совершающих операции над контенйром 1-го уровня:
    - Генерация фигур со случайным значением параметров;
    - Печать контейнера на экран;
    - Удаление элементов со значением площади меньше определенного числа;
- В контенер второго уровня поместить цепочку команд.
- Реализовать цикл, который проходит по всем командам в контенере второго уровня и выполняет их, применяя к контейнеру первого уровня.

Для создания потоков использовать механизмы:

- future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

Нельзя использовать:

- Стандартные контейнеры std.

Код программы:

TBinaryTree.h

```
#ifndef TBINARYTREE_H
#define TBINARYTREE_H
#include "TreeNode.h"
#include <iostream>


template <class TT> class TBinaryTree {
public:
    TBinaryTree();
    void Insert(std::shared_ptr<TT> command);
    void Inorder();

    ~TBinaryTree();
private:
    void Insert(std::shared_ptr<TreeNode<TT>> node, std::shared_ptr<TT>
command);
    void Inorder(std::shared_ptr<TreeNode<TT>> node);

    std::shared_ptr<TreeNode<TT>> root;

};


#define TREE_FUNCTIONS
#include "TBinaryTree.cpp"

#endif
```

TBinaryTree.cpp

```
#ifndef TREE_FUNCTIONS
#include "TBinaryTree.h"

#else

template <class TT> TBinaryTree<TT>::TBinaryTree() {
    root.reset();
}

template <class TT> void TBinaryTree<TT>::Insert(std::shared_ptr<TT> command)
{
    if (root == nullptr) {
        root.reset(new TreeNode<TT>(command));
        return;
    }
    Insert(root, command);
    return;
}

template <class TT> void
TBinaryTree<TT>::Insert(std::shared_ptr<TreeNode<TT>> node,
std::shared_ptr<TT> command) {

    if (node->right == nullptr) {
        node->right.reset(new TreeNode<TT>(command));
```

```
        return;
    }
    Insert(node->right, command);
    return;
}


template <class TT> void TBinaryTree<TT>::Inorder() {
    Inorder(root);
    return;
}

template <class TT> void
TBinaryTree<TT>::Inorder(std::shared_ptr<TreeNode<TT>> node) {
    if (node == nullptr) {
        return;
    }
    Inorder(node->left);
    std::shared_ptr<TT> cmd = node->command;
    (*cmd)();
    Inorder(node->right);
}

template <class TT> TBinaryTree<TT>::~TBinaryTree() {
    root.reset();
}

#endif
```

TreeNode.h

```
#ifndef TREENODE_H
#define TREENODE_H
#include <cstdlib>
#include <memory>

template <class TT> class TreeNode {
public:
    TreeNode() {};
    TreeNode(std::shared_ptr<TT> _command) : command(_command),
left(nullptr), right(nullptr) {};

    std::shared_ptr<TreeNode<TT>> left;
    std::shared_ptr<TreeNode<TT>> right;
    std::shared_ptr<TT> command;

};

#endif
```

main.cpp

```
#include <cstdlib>
#include <iostream>
#include "TQueueItem.h"
#include "TQueue.h"
#include "TBinaryTree.h"
#include <memory>
```

```
#include <functional>
#include <random>

int main(int argc, char** argv) {

    char choice;
    TQueue<Figure> queue;
    typedef std::function<void (void)> command;
    TBinaryTree <command> tree_com;

    command insert = [&]() {
        std::default_random_engine generator;
        std::uniform_int_distribution<int> distribution(1,1000);

        for (int i = 0; i < 5; ++i) {
            int figure = distribution(generator);
            if (figure % 2 == 0) {
                int length = distribution(generator);
                int height = distribution(generator);
                queue.push(std::shared_ptr<Rectangle>(new Rectangle(length,
height)));
            }
            else if (figure % 3 == 1) {
                int up_base = distribution(generator);
                int low_base = distribution(generator);
                int height = distribution(generator);
                queue.push(std::shared_ptr<Trapeze>(new Trapeze(up_base,
low_base, height)));
            }
            else {
                int rhomb_side = distribution(generator);
                int rhomb_height = distribution(generator);
                queue.push(std::shared_ptr<Rhomb>(new Rhomb(rhomb_side,
rhomb_height)));
            }
        }
    };

    command print = [&]() {
        puts("-----------------------------------------------\n");
        for (auto i: queue) {
            i->Print();
            std::cout << '\n';
        }
    };

    command del = [&]() {
        double square = 0;
        std::cout << "Введите значение площади" << std::endl;
        std::cin >> square;
        TQueue<Figure> temp;
        while(!queue.empty()) {
            std::shared_ptr<Figure> figure = queue.pop();
            if (figure->Square() >= square) {
                temp.push(std::move(figure));
            }
        }
        while(!temp.empty()) {
            queue.push(temp.pop());
        }

    };
```

```
    puts("---------------------MENU----------------------\n");
    puts("r - Добавить 5 фигур со случайными параметрами\n");
    puts("           p - Просмотреть очередь            \n");
    puts("            s - Удаление по площади           \n");
    puts("              g - Запустить команды           \n");
    puts("             4 - Удалить очередь              \n");
    puts("                  x - выход                   \n");
    puts("-----------------------------------------------\n");

    do {
        std::cout << '\n';
        std::cin >> choice;

        switch(choice) {
            case 'r': {
                tree_com.Insert(std::shared_ptr<command> (&insert,
[](command*) {}));
                break;
            }
            case 'p':{
                tree_com.Insert(std::shared_ptr<command> (&print,
[](command*) {}));
                break;
            }
            case 's':{
                tree_com.Insert(std::shared_ptr<command> (&del, [](command*)
{}));
                break;
            }
            case 'g':{
                tree_com.Inorder();
                tree_com.~TBinaryTree();
                break;
            }
            case '4':{
                queue.~TQueue();
                break;
            }
            case 'x':{
                break;
            }
            default:{
                std::cout << "Нет варианта" << std::endl;
                break;
            }
        }
    } while (choice != 'x');

    return 0;
}
```

Тестирование:

```
user@lubuntu:~/OOP/lab9$ ./main.exe
---------------------MENU----------------------

r - Добавить 5 фигур со случайными параметрами

           p - Просмотреть очередь
```

```
            s - Удаление по площади

              g - Запустить команды

             4 - Удалить очередь

                  x - выход

-----------------------------------------------


r

p

g
-----------------------------------------------

Трапеция
Верхнее основание: 132
Нижнее основание: 756
Высота: 459
Площадь: 203796

Ромб
Сторона ромба: 219
Высота: 48
Площадь: 10512

Трапеция
Верхнее основание: 680
Нижнее основание: 935
Высота: 384
Площадь: 310080

Прямоугольник
Длина: 831
Высота: 35
Площадь: 29085

Прямоугольник
Длина: 530
Высота: 672
Площадь: 356160


4

r

r

p

s

p

g
-----------------------------------------------

Трапеция
Верхнее основание: 132
```

```
Нижнее основание: 756
Высота: 459
Площадь: 203796

Ромб
Сторона ромба: 219
Высота: 48
Площадь: 10512

Трапеция
Верхнее основание: 680
Нижнее основание: 935
Высота: 384
Площадь: 310080

Прямоугольник
Длина: 831
Высота: 35
Площадь: 29085

Прямоугольник
Длина: 530
Высота: 672
Площадь: 356160

Трапеция
Верхнее основание: 132
Нижнее основание: 756
Высота: 459
Площадь: 203796

Ромб
Сторона ромба: 219
Высота: 48
Площадь: 10512

Трапеция
Верхнее основание: 680
Нижнее основание: 935
Высота: 384
Площадь: 310080

Прямоугольник
Длина: 831
Высота: 35
Площадь: 29085

Прямоугольник
Длина: 530
Высота: 672
Площадь: 356160

Введите значение площади
200000
-----------------------------------------------

Трапеция
Верхнее основание: 132
Нижнее основание: 756
Высота: 459
Площадь: 203796

Трапеция
```

```
Верхнее основание: 680
Нижнее основание: 935
Высота: 384
Площадь: 310080

Прямоугольник
Длина: 530
Высота: 672
Площадь: 356160

Трапеция
Верхнее основание: 132
Нижнее основание: 756
Высота: 459
Площадь: 203796

Трапеция
Верхнее основание: 680
Нижнее основание: 935
Высота: 384
Площадь: 310080

Прямоугольник
Длина: 530
Высота: 672
Площадь: 356160


4

x
user@lubuntu:~/OOP/lab9$
```

Заключение

В ходе данного курса я познакомился с принципом ООП в языке C++ и значительно улучшил свои навыки по написанию классов и использованию объектов этих классов. Каждая лабораторная работа была связана с предыдущей. На протяжении всех лабораторных работ первоначальная программа улучшалась по части безопасности и производительности, в нее вносился новый функционал. В первой лабораторной работе были созданы три класса фигур, была проведена работа по перегрузке операторов, были получены навыки по работе с дружественными функциями, изучены базовые понятия ООП. Во второй лабораторной работе был разработан класс контейнер – очередь, для хранения фигуры, была проведена работа с объектами, передаваемыми по значению. В третьей лабораторной работе были впервые использованы умные указатели, вместо обычных. Это позволило облегчить процесс по выделению и освобождению памяти. В четвертой лабораторной работе была проведена работа с шаблонами классов, был создан шаблон класса-контейнера очередь. В пятой лабораторной работе был разработан итератор для динамической структуры данных, который позволял значительно упростить процесс обхода очереди и вывода всех элементов очереди на экран. В шестой лабораторной работе впервые был реализован контейнер второго уровня – бинарное дерево поиска. На его основе был разработан собственный аллокатор памяти, который позволял значительно ускорить процесс по выделению памяти для объектов классов. Седьмая лабораторная работа показалась самой сложной для меня. Были получены навыки по созданию сложных структур данных, в которых элементом контейнера является другой контейнер, содержащий объекты классов фигур. Был закреплен принцип OCP. В восьмой лабораторной работе были получены навыки по многопоточному программированию на C++, также были разработаны два метода сортировки контейнера, с использованием параллельных вызовов и без их использования. В девятой лабораторной работе я познакомился с лямбда-выражениями и реализовал контейнер второго уровня, который содержал действия над контейнером первого уровня. В итоге выполнения всех лабораторных работ я освоил ряд новых для меня возможностей языка C++, получил хорошие навыки по проектированию и написанию программ, которые могли бы улучшаться и использоваться в последующих проектах.

Ссылка на GitHub: https://github.com/AlexandrFirfarov/OOP