**Московский Авиационный Институт**

**(Национальный Исследовательский Университет)**

Кафедра: 806 «Вычислительная математика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Дисциплина: «Объектно-ориентированное программирование»

Отчёт по лабораторным работам

Вариант 18

Группа: 8О-206Б

Студент: Усков А.С.

Преподаватели:

Дзюба Д.В.

Поповкин А.В.

Москва, 2018

Лабораторная работа №1

Цели:

* Программирование классов на языке С++
* Управление памятью в языке С++
* 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 вариантам заданий.

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).

Код:

Листинг файла Figure.h

#ifndef FIGURE\_H\_INCLUDED

#define FIGURE\_H

class Figure {

public:

virtual double Square() = 0;

virtual void Print() = 0;

virtual ~Figure() {};

};

#endif /\* FIGURE\_H \*/

Листинг файла FourSquare.h

#ifndef SQUARE\_H\_INCLUDED

#define SQUARE\_H\_INCLUDED

#include <cstdlib>

#include <iostream>

#include "Figure.h"

class FourSquare : public Figure {

public:

FourSquare();

FourSquare(std::istream &is);

FourSquare(size\_t a);

FourSquare(const FourSquare& orig);

friend bool operator==(const FourSquare& left,const FourSquare& right);

friend bool operator>(const FourSquare& left, const FourSquare& right);

FourSquare& operator=(const FourSquare& right);

friend FourSquare operator+(const FourSquare& left, const FourSquare& right);

friend std::ostream& operator<<(std::ostream& os, const FourSquare& obj);

friend std::istream& operator>>(std::istream& is, FourSquare& obj);

double Square() override;

void Print() override;

virtual ~FourSquare();

private:

size\_t side;

};

#endif

Листинг файла FourSquare.cpp

#include <iostream>

#include <cmath>

#include "FourSquare.h"

FourSquare::FourSquare(): FourSquare(0) {}

FourSquare::FourSquare(size\_t a):

side(a) {

std::cout << "Foursquare created: " << side << std::endl;

}

FourSquare::FourSquare(std::istream& is) {

is >> side;

std::cout << "FourSquare successfuly read" << std::endl;

}

FourSquare::FourSquare(const FourSquare& orig) {

side = orig.side;

std::cout << "Foursquare successfuly copied" << std::endl;

}

bool operator==(const FourSquare& left, const FourSquare& right) {

return left.side == right.side;

}

bool operator>(const FourSquare& left, const FourSquare& right) {

return left.side > right.side;

}

FourSquare& FourSquare::operator=(const FourSquare& right) {

if (this == &right) return \*this;

std::cout << "Foursquare copied" << std::endl;

this->side = right.side;

return \*this;

}

FourSquare operator+(const FourSquare& left, const FourSquare& right) {

return FourSquare(left.side + right.side);

}

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

os << "Side = " << obj.side << std::endl;

return os;

}

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

is >> obj.side;

return is;

}

double FourSquare::Square() {

return side \* side;

}

void FourSquare::Print() {

std::cout << "Type of figure: Foursquare\nSide = " << side << std::endl;

}

FourSquare::~FourSquare() {

std::cout << "Foursquare deleted" << std::endl;

}

Листинг файла main.cpp

#include <iostream>

#include <cstdlib>

#include "FourSquare.h"

int main() {

Figure\* ptr;

ptr = new FourSquare(2);

ptr->Print();

std::cout << ptr->Square() << std::endl;

delete ptr;

return 0;

}

Выводы:

В ходе выполнения лабораторной̆ работы был получен навык работы с классами в С++. Я научился создавать классы и использовать объекты этих классов. Был спроектирован и запрограммирован на языке С++ класс фигуры: квадрат. Так же были изучены основные понятия ООП и ознакомление с перегрузкой̆ операторов.

Лабораторная работа №2

Цели:

* 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, согласно вариантам заданий (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 соджержать 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).

Код:

Листинг файла TTreeNode.h

#ifndef TTREENODE\_H\_NOT\_INCLUDED

#define TTREENODE\_H\_NOT\_INCLUDED

#include "FourSquare.h"

class TTreeNode {

public:

TTreeNode(const FourSquare& Foursquare);

TTreeNode(const TTreeNode& orig);

friend std::ostream& operator<<(std::ostream& os, const TTreeNode& obj);

friend bool operator>(const TTreeNode& left, const TTreeNode& right);

void SetLeft(TTreeNode\* left);

void SetRight(TTreeNode\* right);

TTreeNode\* GetLeft();

TTreeNode\* GetRight();

FourSquare GetFourSquare() const;

virtual ~TTreeNode();

private:

FourSquare Foursquare;

TTreeNode\* left;

TTreeNode\* right;

};

#endif

Листинг файла TTreeNode.cpp

#include <iostream>

#include "TTreeNode.h"

TTreeNode::TTreeNode(const FourSquare& Foursquare) {

this->Foursquare = Foursquare;

this->left = nullptr;

this->right = nullptr;

std::cout << "Tree node: created" << std::endl;

}

TTreeNode::TTreeNode(const TTreeNode& orig) {

this->Foursquare = orig.Foursquare;

this->left = orig.left;

this->right = orig.right;

std::cout << "Tree node: copied" << std::endl;

}

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

os << "[" << obj.Foursquare << "]" << std::endl;

return os;

}

bool operator>(const TTreeNode& left, const TTreeNode& right) {

return left.GetFourSquare() > right.GetFourSquare();

}

void TTreeNode::SetLeft(TTreeNode\* left) {

this->left = left;

}

void TTreeNode::SetRight(TTreeNode\* right) {

this->right = right;

}

TTreeNode\* TTreeNode::GetLeft() {

return this->left;

}

TTreeNode\* TTreeNode::GetRight() {

return this->right;

}

FourSquare TTreeNode::GetFourSquare() const {

return this->Foursquare;

}

TTreeNode::~TTreeNode() {

std::cout << "Tree node: deleted" << std::endl;

delete left;

delete right;

}

Листинг файла TTree.h

#ifndef TTREE\_H\_NOT\_INCLUDED

#define TTREE\_H\_NOT\_INCLUDED

#include "TTreeNode.h"

#include "FourSquare.h"

class TTree {

public:

TTree();

TTree(const TTree& orig);

void add(FourSquare& Foursquare);

bool empty();

void del(FourSquare& Foursquare);

friend std::ostream& operator<<(std::ostream& os, const TTree& tree);

virtual ~TTree();

private:

TTreeNode\* root;

};

#endif

Листинг файла TTree.cpp

#include <iostream>

#include "TTree.h"

TTree::TTree() {

this->root = nullptr;

}

TTree::TTree(const TTree& tree) {

this->root = tree.root;

}

void TTree::add(Figure& Fig) {

TTreeNode\* father = nullptr;

TTreeNode\* curr = this->root;

while (curr != nullptr) {

father = curr;

if (curr->GetFigure()->Type() > Fig->Type() ||(curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() > Fig->Square())) {

curr = curr->GetLeft();

} else if (curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() == Fig->Square()) {

std::cout << "Already in tree" << std::endl;

return;

} else {

curr = curr->GetRight();

}

}

curr = (new TTreeNode<T>(Fig));

if (this->root == nullptr) this->root = curr;

else if (father->GetFigure()->Type() > Fig->Type() || (father->GetFigure()->Type() == Fig->Type() && father->GetFigure()->Square() > Fig->Square())) father->SetLeft(curr);

else father->SetRight(curr);

curr->SetParent(father);

return;

}

bool TTree::empty() {

return this->root == nullptr;

}

void TTree::del(Figure& Fig)) {

TTreeNode\* curr = this->root;

TTreeNode\* father = nullptr;

while (curr != nullptr && (curr->GetFigure()->Type() != Fig->Type() || curr->GetFigure()->Square() != Fig->Square())) {

father = curr;

if (curr->GetFigure()->Type() > Fig->Type() || (curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() > Fig->Square())) curr = curr->GetLeft();

else curr = curr->GetRight();

}

if (curr == nullptr) {

std::cout << "Not in tree" << std::endl;

return;

}

if (curr->GetLeft() == nullptr && curr->GetRight() == nullptr) {

if (curr == this->root) {

this->root = nullptr;

return;

}

if (father->GetLeft() == curr) father->SetLeft(nullptr);

else father->SetRight(nullptr);

curr->SetParent(nullptr);

curr = nullptr;

return;

}

if (curr->GetRight() == nullptr) {

if (father == nullptr) {

this->root = curr->GetLeft();

} else if (father->GetFigure()->Type() > curr->GetFigure()->Type() || (father->GetFigure()->Type() == curr->GetFigure()->Type() && father->GetFigure()->Square() > curr->GetFigure()->Square())) {

father->SetLeft(curr->GetLeft());

} else {

father->SetRight(curr->GetLeft());

}

curr->GetLeft()->SetParent(father);

curr->SetLeft(nullptr);

curr = nullptr;

return;

}

if (curr->GetLeft() == nullptr) {

if (father == nullptr) {

this->root = curr->GetRight();

} else if (father->GetFigure()->Type() > curr->GetFigure()->Type() || (father->GetFigure()->Type() == curr->GetFigure()->Type() && father->GetFigure()->Square() > curr->GetFigure()->Square())) {

father->SetLeft(curr->GetRight());

} else {

father->SetRight(curr->GetRight());

}

curr->GetRight()->SetParent(father);

curr->SetRight(nullptr);

curr = nullptr;

return;

}

TTreeNode\* currd = curr->GetRight();

TTreeNode\* prev = curr;

while (currd->GetLeft() != nullptr) {

prev = currd;

currd = currd->GetLeft();

}

curr->SetFigure(currd->GetFigure());

if (prev == curr) {

if (currd->GetRight() != nullptr) currd->GetRight()->SetParent(curr);

curr->SetRight(currd->GetRight());

} else {

if (currd->GetRight() != nullptr) currd->GetRight()->SetParent(prev);

prev->SetLeft(currd->GetRight());

}

currd->SetRight(nullptr);

currd->SetParent(nullptr);

Delete currd;

return;

}

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

if (obj.root == nullptr) return os;

os << obj.root->GetLeft() << std::endl;

os << obj.root->GetFourSquare() << std::endl;

os << obj.root->GetRight() << std::endl;

}

TTree::~TTree() {

Delete this->root;

}

Выводы:

В ходе этой лабораторной работы были закреплены навыки работы с классами. Повторены способы создания и использования объектов этих классов. Так же я ознакомился с динамическими структурами. Были написаны динамические структуры, работающие с объектами, передаваемыми “по значению”.

Лабораторная работа №3

Цели:

* 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 вариантам заданий (реализованную в ЛР1).

Классы должны удовлетворять следующим правилам:

* Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
* Класс-контейнер должен содержать 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 (определяется структурой контейнера).
* Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется 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 расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Код:

Листинг Rectangle.h

#ifndef RECTANGLE\_H\_INCLUDED

#define RECTANGLE\_H\_INCLUDED

#include <cstdlib>

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure {

public:

Rectangle();

Rectangle(std::istream &is);

Rectangle(size\_t a, size\_t b);

Rectangle(const Rectangle& orig);

Rectangle& operator=(const Rectangle& right);

friend Rectangle operator+(const Rectangle& left, const Rectangle& right);

friend std::ostream& operator<<(std::ostream& os, const Rectangle& obj);

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

double Square() const override;

size\_t Type() const override;

std::ostream& Print(std::ostream& os) const override;

std::istream& Scan(std::istream& is) override;

virtual ~Rectangle();

private:

size\_t side\_a;

size\_t side\_b;

size\_t typ;

};

#endif

Листинг Rectangle.cpp

#include <iostream>

#include <cmath>

#include "Rectangle.h"

Rectangle::Rectangle(): Rectangle(0, 0) {}

Rectangle::Rectangle(size\_t a, size\_t b) :

side\_a(a), side\_b(b) {

this->typ = 1;

//std::cout << "Rectangle created: " << side\_a << ", " << side\_b << std::endl;

}

Rectangle::Rectangle(std::istream& is) {

is >> side\_a;

is >> side\_b;

typ = 1;

//std::cout << "Rectangle successfuly read" << std::endl;

}

Rectangle::Rectangle(const Rectangle& orig) {

side\_a = orig.side\_a;

side\_b = orig.side\_b;

typ = 1;

//std::cout << "Rectangle successfuly copied" << std::endl;

}

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

if (this == &right) return \*this;

//std::cout << "Foursquare copied" << std::endl;

this->side\_a = right.side\_a;

this->side\_b = right.side\_b;

return \*this;

}

Rectangle operator+(const Rectangle& left, const Rectangle& right) {

return Rectangle(left.side\_a + right.side\_a, left.side\_b + right.side\_b);

}

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

os << "Type of figure: Rectangle\nSides = " << obj.side\_a << ", " << obj.side\_b << std::endl;

return os;

}

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

obj = Rectangle(is);

return is;

}

double Rectangle::Square() const {

return side\_a \* side\_b;

}

size\_t Rectangle::Type() const {

return this->typ;

}

std::ostream& Rectangle::Print(std::ostream& os) const {

os << \*this;

return os;

}

std::istream& Rectangle::Scan(std::istream& is) {

is >> \*this;

return is;

}

Rectangle::~Rectangle() {

//std::cout << "Rectangle deleted" << std::endl;

}

Листинг Trapeze.h

#ifndef TRAPEZE\_H\_INCLUDED

#define TRAPEZE\_H\_INCLUDED

#include <cstdlib>

#include <iostream>

#include "Figure.h"

class Trapeze : public Figure {

public:

Trapeze();

Trapeze(std::istream &is);

Trapeze(size\_t a, size\_t b, size\_t h);

Trapeze(const Trapeze& orig);

Trapeze& operator=(const Trapeze& right);

friend Trapeze operator+(const Trapeze& left, const Trapeze& right);

friend std::ostream& operator<<(std::ostream& os, const Trapeze& obj);

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

double Square() const override;

size\_t Type() const override;

std::ostream& Print(std::ostream& os) const override;

std::istream& Scan(std::istream& is) override;

virtual ~Trapeze();

private:

size\_t basis\_a;

size\_t basis\_b;

size\_t height;

size\_t typ;

};

#endif

Листинг Trapeze.cpp

#include <iostream>

#include <cmath>

#include "Rectangle.h"

Rectangle::Rectangle(): Rectangle(0, 0) {}

Rectangle::Rectangle(size\_t a, size\_t b) :

side\_a(a), side\_b(b) {

this->typ = 1;

//std::cout << "Rectangle created: " << side\_a << ", " << side\_b << std::endl;

}

Rectangle::Rectangle(std::istream& is) {

is >> side\_a;

is >> side\_b;

typ = 1;

//std::cout << "Rectangle successfuly read" << std::endl;

}

Rectangle::Rectangle(const Rectangle& orig) {

side\_a = orig.side\_a;

side\_b = orig.side\_b;

typ = 1;

//std::cout << "Rectangle successfuly copied" << std::endl;

}

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

if (this == &right) return \*this;

//std::cout << "Foursquare copied" << std::endl;

this->side\_a = right.side\_a;

this->side\_b = right.side\_b;

return \*this;

}

Rectangle operator+(const Rectangle& left, const Rectangle& right) {

return Rectangle(left.side\_a + right.side\_a, left.side\_b + right.side\_b);

}

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

os << "Type of figure: Rectangle\nSides = " << obj.side\_a << ", " << obj.side\_b << std::endl;

return os;

}

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

obj = Rectangle(is);

return is;

}

double Rectangle::Square() const {

return side\_a \* side\_b;

}

size\_t Rectangle::Type() const {

return this->typ;

}

std::ostream& Rectangle::Print(std::ostream& os) const {

os << \*this;

return os;

}

std::istream& Rectangle::Scan(std::istream& is) {

is >> \*this;

return is;

}

Rectangle::~Rectangle() {

//std::cout << "Rectangle deleted" << std::endl;

}

Листинг файла TTreeNode.h

#ifndef TTREENODE\_H\_NOT\_INCLUDED

#define TTREENODE\_H\_NOT\_INCLUDED

#include "FourSquare.h"

class TTreeNode {

public:

TTreeNode(const FourSquare& Foursquare);

TTreeNode(const TTreeNode& orig);

friend std::ostream& operator<<(std::ostream& os, const TTreeNode& obj);

friend bool operator>(const TTreeNode& left, const TTreeNode& right);

void SetLeft(TTreeNode\* left);

void SetRight(TTreeNode\* right);

TTreeNode\* GetLeft();

TTreeNode\* GetRight();

FourSquare GetFourSquare() const;

virtual ~TTreeNode();

private:

FourSquare Foursquare;

TTreeNode\* left;

TTreeNode\* right;

};

#endif

Листинг файла TTreeNode.h

#ifndef TTREENODE\_H\_INCLUDED

#define TTREENODE\_H\_INCLUDED

#include <iostream>

#include <cstdlib>

#include <memory>

#include "TStack.h"

#include "Figure.h"

class TTreeNode {

public:

TTreeNode(const std::shared\_ptr<Figure>& Fig) {

this->Fig = Fig;

this->left = nullptr;

this->right = nullptr;

this->parent = nullptr;

//std::cout << "Tree node: created" << std::endl;

}

TTreeNode(const std::shared\_ptr<TTreeNode>& orig) {

this->Fig = orig->Fig;

this->left = orig->left;

this->right = orig->right;

this->parent = orig->parent;

//std::cout << "Tree node: copied" << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, const TTreeNode& obj) {

if (obj.left != nullptr) os << \*obj.left;

os << obj.Fig;

if (obj.right != nullptr) os << \*obj.right;

return os;

}

friend bool operator>(const std::shared\_ptr<TTreeNode>& left, const std::shared\_ptr<TTreeNode>& right) {

return left->Fig > right->Fig;

}

void SetLeft(std::shared\_ptr<TTreeNode> left) {

this->left = left;

}

void SetRight(std::shared\_ptr<TTreeNode> right) {

this->right = right;

}

void SetParent(std::shared\_ptr<TTreeNode> parent) {

this->parent = parent;

}

void SetFigure(const std::shared\_ptr& Fig) {

this->Fig = Fig;

}

std::shared\_ptr<TTreeNode> GetLeft() {

if (this == nullptr) return std::shared\_ptr<TTreeNode>(this);

return this->left;

}

std::shared\_ptr<TTreeNode> GetRight() {

if (this == nullptr) return std::shared\_ptr<TTreeNode>(this);

return this->right;

}

std::shared\_ptr<TTreeNode> GetParent() {

return this->parent;

}

std::shared\_ptr<T> GetFigure() {

return this->Fig;

}

virtual ~TTreeNode() {

//std::cout << "Tree node: deleted" << std::endl;

}

private:

Figure Fig;

std::shared\_ptr<TTreeNode> left;

std::shared\_ptr<TTreeNode> right;

std::shared\_ptr<TTreeNode> parent;

};

#endif

Листинг файла TTree.h

#ifndef TTREE\_H\_INCLUDED

#define TTREE\_H\_INCLUDED

#include <iostream>

#include <memory>

#include "TTreeNode.h"

#include "Figure.h"

template <class T>

class TTree {

public:

TTree() {

this->root = nullptr;

}

TTree(const TTree& orig) {

this->root = orig.root;

}

std::shared\_ptr<TTreeNode> GetRoot() {

return this->root;

}

void add(std::shared\_ptr<Figure>& Fig) {

std::shared\_ptr<TTreeNode> father = nullptr;

std::shared\_ptr<TTreeNode> curr = this->root;

while (curr != nullptr) {

father = curr;

if (curr->GetFigure()->Type() > Fig->Type() ||(curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() > Fig->Square())) {

curr = curr->GetLeft();

} else if (curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() == Fig->Square()) {

std::cout << "Already in tree" << std::endl;

return;

} else {

curr = curr->GetRight();

}

}

curr = std::shared\_ptr<TTreeNode>(new TTreeNode<T>(Fig));

if (this->root == nullptr) this->root = curr;

else if (father->GetFigure()->Type() > Fig->Type() || (father->GetFigure()->Type() == Fig->Type() && father->GetFigure()->Square() > Fig->Square())) father->SetLeft(curr);

else father->SetRight(curr);

curr->SetParent(father);

return;

}

bool empty() {

return this->root == nullptr;

}

void del(std::shared\_ptr<T>& Fig) {

std::shared\_ptr<TTreeNode> curr = this->root;

std::shared\_ptr<TTreeNode> father = nullptr;

while (curr != nullptr && (curr->GetFigure()->Type() != Fig->Type() || curr->GetFigure()->Square() != Fig->Square())) {

father = curr;

if (curr->GetFigure()->Type() > Fig->Type() || (curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() > Fig->Square())) curr = curr->GetLeft();

else curr = curr->GetRight();

}

if (curr == nullptr) {

std::cout << "Not in tree" << std::endl;

return;

}

if (curr->GetLeft() == nullptr && curr->GetRight() == nullptr) {

if (curr == this->root) {

this->root = nullptr;

return;

}

if (father->GetLeft() == curr) father->SetLeft(nullptr);

else father->SetRight(nullptr);

curr->SetParent(nullptr);

curr = nullptr;

return;

}

if (curr->GetRight() == nullptr) {

if (father == nullptr) {

this->root = curr->GetLeft();

} else if (father->GetFigure()->Type() > curr->GetFigure()->Type() || (father->GetFigure()->Type() == curr->GetFigure()->Type() && father->GetFigure()->Square() > curr->GetFigure()->Square())) {

father->SetLeft(curr->GetLeft());

} else {

father->SetRight(curr->GetLeft());

}

curr->GetLeft()->SetParent(father);

curr->SetLeft(nullptr);

curr = nullptr;

return;

}

if (curr->GetLeft() == nullptr) {

if (father == nullptr) {

this->root = curr->GetRight();

} else if (father->GetFigure()->Type() > curr->GetFigure()->Type() || (father->GetFigure()->Type() == curr->GetFigure()->Type() && father->GetFigure()->Square() > curr->GetFigure()->Square())) {

father->SetLeft(curr->GetRight());

} else {

father->SetRight(curr->GetRight());

}

curr->GetRight()->SetParent(father);

curr->SetRight(nullptr);

curr = nullptr;

return;

}

std::shared\_ptr<TTreeNode> currd = curr->GetRight();

std::shared\_ptr<TTreeNode> prev = curr;

while (currd->GetLeft() != nullptr) {

prev = currd;

currd = currd->GetLeft();

}

curr->SetFigure(currd->GetFigure());

if (prev == curr) {

if (currd->GetRight() != nullptr) currd->GetRight()->SetParent(curr);

curr->SetRight(currd->GetRight());

} else {

if (currd->GetRight() != nullptr) currd->GetRight()->SetParent(prev);

prev->SetLeft(currd->GetRight());

}

currd->SetRight(nullptr);

currd->SetParent(nullptr);

currd = nullptr;

return;

}

friend std::ostream& operator<<(std::ostream& os, const TTree& tree) {

if (tree.root == nullptr) {

std::cout << "Tree is empty" << std::endl;

return os;

}

os << \*tree.root;

return os;

}

virtual ~TTree() {

this->root = nullptr;

std::cout << "Tree deleted" << std::endl;

}

private:

std::shared\_ptr<TTreeNode> root;

};

#endif

Выводы:

В ходе выполнения данной лабораторной работы были окончательно закреплены навыки работы с классами и был получен навык использования “умных” указателей. Были спроектированы и запрограммированы на языке С++ классы фигур: квадрат, прямоугольник и трапеция. Все программы проекта были переписаны на использование “умных” указателей.

Лабораторная работа №4

Цели:

* 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 вариантам заданий (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 (опеределяется структурой контейнера).
* Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (опеределяется 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).

Код:

Листинг файла TTreeNode.h

#ifndef TTREENODE\_H\_INCLUDED

#define TTREENODE\_H\_INCLUDED

#include <iostream>

#include <cstdlib>

#include <memory>

#include "TStack.h"

#include "Figure.h"

template <class T>

class TTreeNode {

public:

TTreeNode(const std::shared\_ptr<T>& Fig) {

this->Fig = Fig;

this->left = nullptr;

this->right = nullptr;

this->parent = nullptr;

//std::cout << "Tree node: created" << std::endl;

}

TTreeNode(const std::shared\_ptr<TTreeNode<T>>& orig) {

this->Fig = orig->Fig;

this->left = orig->left;

this->right = orig->right;

this->parent = orig->parent;

//std::cout << "Tree node: copied" << std::endl;

}

template <class A>

friend std::ostream& operator<<(std::ostream& os, const TTreeNode<A>& obj) {

if (obj.left != nullptr) os << \*obj.left;

os << obj.Fig;

if (obj.right != nullptr) os << \*obj.right;

return os;

}

friend bool operator>(const std::shared\_ptr<TTreeNode<T>>& left, const std::shared\_ptr<TTreeNode<T>>& right) {

return left->Fig > right->Fig;

}

void SetLeft(std::shared\_ptr<TTreeNode<T>> left) {

this->left = left;

}

void SetRight(std::shared\_ptr<TTreeNode<T>> right) {

this->right = right;

}

void SetParent(std::shared\_ptr<TTreeNode<T>> parent) {

this->parent = parent;

}

void SetFigure(const std::shared\_ptr<T>& Fig) {

this->Fig = Fig;

}

std::shared\_ptr<TTreeNode<T>> GetLeft() {

if (this == nullptr) return std::shared\_ptr<TTreeNode<T>>(this);

return this->left;

}

std::shared\_ptr<TTreeNode<T>> GetRight() {

if (this == nullptr) return std::shared\_ptr<TTreeNode<T>>(this);

return this->right;

}

std::shared\_ptr<TTreeNode<T>> GetParent() {

return this->parent;

}

std::shared\_ptr<T> GetFigure() {

return this->Fig;

}

virtual ~TTreeNode() {

//std::cout << "Tree node: deleted" << std::endl;

}

private:

std::shared\_ptr<T> Fig;

std::shared\_ptr<TTreeNode<T>> left;

std::shared\_ptr<TTreeNode<T>> right;

std::shared\_ptr<TTreeNode<T>> parent;

};

#endif

Листинг файла TTree.h

#ifndef TTREE\_H\_INCLUDED

#define TTREE\_H\_INCLUDED

#include <iostream>

#include <memory>

#include "TTreeNode.h"

#include "Figure.h"

template <class T>

class TTree {

public:

TTree() {

this->root = nullptr;

}

TTree(const TTree& orig) {

this->root = orig.root;

}

std::shared\_ptr<TTreeNode<T>> GetRoot() {

return this->root;

}

void add(std::shared\_ptr<T>& Fig) {

std::shared\_ptr<TTreeNode<T>> father = nullptr;

std::shared\_ptr<TTreeNode<T>> curr = this->root;

while (curr != nullptr) {

father = curr;

if (curr->GetFigure()->Type() > Fig->Type() ||(curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() > Fig->Square())) {

curr = curr->GetLeft();

} else if (curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() == Fig->Square()) {

std::cout << "Already in tree" << std::endl;

return;

} else {

curr = curr->GetRight();

}

}

curr = std::shared\_ptr<TTreeNode<T>>(new TTreeNode<T>(Fig));

if (this->root == nullptr) this->root = curr;

else if (father->GetFigure()->Type() > Fig->Type() || (father->GetFigure()->Type() == Fig->Type() && father->GetFigure()->Square() > Fig->Square())) father->SetLeft(curr);

else father->SetRight(curr);

curr->SetParent(father);

return;

}

bool empty() {

return this->root == nullptr;

}

void del(std::shared\_ptr<T>& Fig) {

std::shared\_ptr<TTreeNode<T>> curr = this->root;

std::shared\_ptr<TTreeNode<T>> father = nullptr;

while (curr != nullptr && (curr->GetFigure()->Type() != Fig->Type() || curr->GetFigure()->Square() != Fig->Square())) {

father = curr;

if (curr->GetFigure()->Type() > Fig->Type() || (curr->GetFigure()->Type() == Fig->Type() && curr->GetFigure()->Square() > Fig->Square())) curr = curr->GetLeft();

else curr = curr->GetRight();

}

if (curr == nullptr) {

std::cout << "Not in tree" << std::endl;

return;

}

if (curr->GetLeft() == nullptr && curr->GetRight() == nullptr) {

if (curr == this->root) {

this->root = nullptr;

return;

}

if (father->GetLeft() == curr) father->SetLeft(nullptr);

else father->SetRight(nullptr);

curr->SetParent(nullptr);

curr = nullptr;

return;

}

if (curr->GetRight() == nullptr) {

if (father == nullptr) {

this->root = curr->GetLeft();

} else if (father->GetFigure()->Type() > curr->GetFigure()->Type() || (father->GetFigure()->Type() == curr->GetFigure()->Type() && father->GetFigure()->Square() > curr->GetFigure()->Square())) {

father->SetLeft(curr->GetLeft());

} else {

father->SetRight(curr->GetLeft());

}

curr->GetLeft()->SetParent(father);

curr->SetLeft(nullptr);

curr = nullptr;

return;

}

if (curr->GetLeft() == nullptr) {

if (father == nullptr) {

this->root = curr->GetRight();

} else if (father->GetFigure()->Type() > curr->GetFigure()->Type() || (father->GetFigure()->Type() == curr->GetFigure()->Type() && father->GetFigure()->Square() > curr->GetFigure()->Square())) {

father->SetLeft(curr->GetRight());

} else {

father->SetRight(curr->GetRight());

}

curr->GetRight()->SetParent(father);

curr->SetRight(nullptr);

curr = nullptr;

return;

}

std::shared\_ptr<TTreeNode<T>> currd = curr->GetRight();

std::shared\_ptr<TTreeNode<T>> prev = curr;

while (currd->GetLeft() != nullptr) {

prev = currd;

currd = currd->GetLeft();

}

curr->SetFigure(currd->GetFigure());

if (prev == curr) {

if (currd->GetRight() != nullptr) currd->GetRight()->SetParent(curr);

curr->SetRight(currd->GetRight());

} else {

if (currd->GetRight() != nullptr) currd->GetRight()->SetParent(prev);

prev->SetLeft(currd->GetRight());

}

currd->SetRight(nullptr);

currd->SetParent(nullptr);

currd = nullptr;

return;

}

template<class A>

friend std::ostream& operator<<(std::ostream& os, const TTree<A>& tree) {

if (tree.root == nullptr) {

std::cout << "Tree is empty" << std::endl;

return os;

}

os << \*tree.root;

return os;

}

virtual ~TTree() {

this->root = nullptr;

std::cout << "Tree deleted" << std::endl;

}

private:

std::shared\_ptr<TTreeNode<T>> root;

};

#endif

Листинг файла Source.cpp

#include <iostream>

#include <cstdlib>

#include <memory>

#include "TTreeNode.h"

#include "TTree.h"

#include "Figure.h"

#include "FourSquare.h"

#include "Rectangle.h"

#include "Trapeze.h"

void Help() {

std::cout << "To print tree enter: p" << std::endl;

std::cout << "To add figure to tree enter: a [name\_of\_figure] [side]" << std::endl;

std::cout << "To delete figure from tree enter: d [name\_of\_figure] [side]" << std::endl;

std::cout << "To exit programm enter: e" << std::endl;

std::cout << "To see this messege again enter: h\n" << std::endl;

std::cout << "Names of figures:\ns - square\nr - rectangle\nt - trapeze\n" << std::endl;

}

int main() {

TTree<Figure> T;

Help();

char ch;

std::shared\_ptr<Figure> ptr;

while (true) {

std::cin >> ch;

if (ch == 'e') {

break;

} else if (ch == 'a') {

std::cin >> ch;

if (ch == 's') ptr = std::shared\_ptr<Figure>(new FourSquare(std::cin));

else if (ch == 'r') ptr = std::shared\_ptr<Figure>(new Rectangle(std::cin));

else if (ch == 't') ptr = std::shared\_ptr<Figure>(new Trapeze(std::cin));

else {

std::cout << "Wrong figure" << std::endl;

continue;

}

T.add(ptr);

} else if (ch == 'd') {

std::cin >> ch;

if (ch == 's') ptr = std::shared\_ptr<Figure>(new FourSquare(std::cin));

else if (ch == 'r') ptr = std::shared\_ptr<Figure>(new Rectangle(std::cin));

else if (ch == 't') ptr = std::shared\_ptr<Figure>(new Trapeze(std::cin));

else {

std::cout << "Wrong figure" << std::endl;

continue;

}

T.del(ptr);

} else if (ch == 'p') {

std::cout << T << std::endl;

} else if (ch == 'h') {

Help();

} else {

continue;

}

}

return 0;

}

Выводы:

В ходе выполнения лабораторной̆ работы был получен навык работы с шаблонами в С++. Я научился создавать шаблоны классов и использовать их. Была спроектирована и запрограммирована на языке С++ динамическая структура, использующая шаблоны ранее запрограммированных классов.

Лабораторная работа №5

Цели:

* 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;

Код:

Листинг файла TIterator.cpp

#ifndef TITERATOR\_H\_INCLUDED

#define TITERATOR\_H\_INCLUDED

#include <iostream>

#include <memory>

template <class TNode, class T>

class TIterator {

public:

TIterator(std::shared\_ptr<TNode> n) {

node\_ptr = n;

}

std::shared\_ptr<T> operator\* () {

return node\_ptr->GetValue();

}

std::shared\_ptr<T> operator-> () {

return node\_ptr->GetValue();

}

void operator-- () {

node\_ptr = node\_ptr->GetPrev();

}

TIterator operator-- (int) {

TIterator iter(\*this);

--(\*this);

return iter;

}

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 !(\*this == i);

}

private:

std::shared\_ptr <TNode> node\_ptr;

};

#endif

Выводы:

В ходе выполнения лабораторной̆ работы был получен навык работы с итераторами в С++. Я научился создавать итераторы классов и использовать итераторы объектов этих классов. Были спроектированы и запрограммированы на языке С++ итераторы классов фигур: квадрат, прямоугольник и трапеция. Также были закреплены основы работы с шаблонами.

Лабораторная работа №6

Цели:

* 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 вариантам заданий).

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

Код:

Листинг файла TStack.h

#ifndef TSTACK\_H\_INCLUDED

#define TSTACK\_H\_INCLUDED

#include <cstdlib>

#include "TStackItem.h"

class TStack {

public:

TStack();

TStack(const TStack& orig);

void push(void\* obj);

bool empty();

void\* pop();

virtual ~TStack();

private:

TStackItem \*head;

};

#endif

Листинг файла TStack.cpp

#include <iostream>

#include "TStack.h"

TStack::TStack() {

head = nullptr;

}

TStack::TStack(const TStack& orig) {

head = orig.head;

}

void TStack::push(void\* obj) {

TStackItem \*other = new TStackItem(obj);

other->SetNext(head);

head = other;

}

bool TStack::empty() {

return head == nullptr;

}

void\* TStack::pop() {

void\* result = nullptr;

if (head != nullptr) {

TStackItem \*old\_head = head;

head = head->GetNext();

result = old\_head->GetPointer();

old\_head->SetNext(nullptr);

delete old\_head;

}

return result;

}

TStack::~TStack() {

delete this->head;

}

Листинг файла TStackItem.h

#ifndef TSTACKITEM\_H\_INCLUDED

#define TSTACKITEM\_H\_INCLUDED

class TStackItem {

public:

TStackItem(void\* obj);

TStackItem(const TStackItem& orig);

TStackItem\* SetNext(TStackItem\* next);

TStackItem\* GetNext();

void\* GetPointer() const;

virtual ~TStackItem();

private:

TStackItem\* next;

void\* pointer;

};

#endif

Листинг файла TStackItem.cpp

#include <iostream>

#include "TStackItem.h"

TStackItem::TStackItem(void\* obj) {

pointer = obj;

next = nullptr;

}

TStackItem::TStackItem(const TStackItem& orig) {

pointer = orig.pointer;

next = orig.next;

}

TStackItem\* TStackItem::SetNext(TStackItem\* next) {

TStackItem\* old = this->next;

this->next = next;

return old;

}

TStackItem\* TStackItem::GetNext() {

return this->next;

}

void\* TStackItem::GetPointer() const {

return this->pointer;

}

TStackItem::~TStackItem() {

delete this->next;

this->pointer = nullptr;

}

Листинг файла TAllocationBlock.h

#ifndef TALLOCATIONBLOCK\_H\_INCLUDED

#define TALLOCATIONBLOCK\_H\_INCLUDED

#include <cstdlib>

#include "TStack.h"

class TAllocationBlock {

public:

TAllocationBlock(size\_t size, size\_t count) {

\_used\_blocks = (char\*)malloc(\_size \* \_count);

\_free\_blocks = new TStack();

for (size\_t i = 0; i < \_count; ++i) \_free\_blocks->push(\_used\_blocks + i \* \_size);

\_free\_count = \_count;

std::cout << "TAllocationBlock: Memory init" << std::endl;

}

void \*allocate() {

void \*result = nullptr;

if (\_free\_count > 0) {

result = \_free\_blocks->pop();

\_free\_count--;

std::cout << "TAllocationBlock: Allocate " << (\_count - \_free\_count) << " of " << \_count << std::endl;

}

else {

std::cout << "TAllocationBlock: No memory exception" << std::endl;

}

return result;

}

void deallocate(void \*pointer) {

std::cout << "TAllocationBlock: Deallocate block " << std::endl;

\_free\_blocks->push(pointer);

\_free\_count++;

}

bool has\_free\_blocks() {

return \_free\_count > 0;

}

~TAllocationBlock() {

if (\_free\_count < \_count) std::cout << "TAllocationBlock: Memory leak?" << std::endl;

else std::cout << "TAllocationBlock: Memory freed" << std::endl;

delete \_free\_blocks;

}

private:

size\_t \_size;

size\_t \_count;

char \*\_used\_blocks;

TStack\* \_free\_blocks;

size\_t \_free\_count;

};

#endif

Выводы:

В ходе выполнения лабораторной̆ работы был получен навык работы с аллокаторами классов в С++. Я научился писать алкллокаторы классов. Был спроектирован и запрограммирован на языке С++ аллокатор динамического класса, использующий другой динамический класс в основе. Так же были закреплены основные понятия о работе с памятью в C++.

Лабораторная работа №7

Цели:

* Создание сложных динамических структур данных.
* Закрепление принципа ООП.

Задание:

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

1. Массив
2. Связный̆ 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 динамической̆ структурой̆ данных одного из следующих видов (Контейнер 2-го уровня):

1. Массив
2. Связный̆ 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.

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

При этом должно выполняться правило, что количество объектов в контейнере второго уровня не больше 5. 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.

Код:

Выводы:

В ходе выполнения лабораторной̆ работы был значительно повышен навык программирования динамических структур. Была спроектирована и написана динамическая структура, имеющая внутри себя объекты другой динамической структуры. Были закреплены основные принципы ООП.

Лабораторная работа №8

Цели:

* 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.

Код:

Листинг файла TTree.cpp

#ifndef TTREE\_H\_INCLUDED

#define TTREE\_H\_INCLUDED

#include <iostream>

#include <memory>

#include "TIterator.h"

#include "TTreeNode.h"

class TTree {

public:

TTree() {

this->Root = nullptr;

}

void Insert (std::shared\_ptr<MyFigure> new\_elem) {

std::shared\_ptr<TTreeNode> father = std::shared\_ptr<TTreeNode>(nullptr);

std::shared\_ptr<TTreeNode> curr = this->Root;

while (curr != nullptr) {

curr->GetValue()->Insert(new\_elem);

if (curr->GetValue()->GetSize() <= 5) return;

new\_elem = curr->GetValue()->Pop();

father = curr;

if (curr->GetValue()->Top() < new\_elem) {

if (!curr->RightIns()) curr = curr->GetRight();

else curr = nullptr;

} else if (curr->GetValue()->Find(new\_elem) != nullptr) {

std::cout << "Already in tree" << std::endl;

return;

} else {

if (!curr->LeftIns()) curr = curr->GetLeft();

else curr = nullptr;

}

}

curr = std::shared\_ptr<TTreeNode>(new TTreeNode(new\_elem));

if (this->Root == nullptr) {

this->Root = curr;

this->Root->InsRight();

this->Root->InsLeft();

} else if (father->GetValue()->GetSize() < 5) {

father->GetValue()->Insert(new\_elem);

} else if (father->GetValue()->Top() > new\_elem) {

father->UnInsLeft();

curr->InsLeft();

curr->SetLeft(father->GetLeft());

father->SetLeft(curr);

curr->InsRight();

curr->SetRight(father);

} else {

curr->InsRight();

curr->SetRight(father->GetRight());

father->SetRight(curr);

curr->InsLeft();

curr->SetLeft(father);

}

std::cout << "Inserted" << std::endl;

return;

}

void Delete (std::shared\_ptr<MyFigure> del\_elem) {

std::shared\_ptr<TTreeNode> curr = this->Root;

std::shared\_ptr<TTreeNode> father = nullptr;

while (curr != nullptr && curr->GetValue()->Find(del\_elem) == nullptr){

father = curr;

if (curr->GetValue()->Top() < del\_elem) curr = !curr->RightIns() ? curr->GetRight() : nullptr;

else if (curr->GetValue()->Find(del\_elem) != nullptr) break;

else curr = !curr->LeftIns() ? curr->GetLeft() : nullptr;

}

if (curr == nullptr) {

std::cout << "Not in tree" << std::endl;

return;

}

curr->GetValue()->Delete(del\_elem);

if (!curr->GetValue()->Empty()) return;

if (!curr->LeftIns() && !curr->RightIns()) {

if (curr == this->Root) this->Root = nullptr;

else if (father->GetRight() == curr) {

father->SetRight(curr->GetRight());

father->InsRight();

} else {

father->SetLeft(curr->GetLeft());

father->InsLeft();

}

return;

}

if (!curr->RightIns()) {

if (father == nullptr) this->Root = curr->GetLeft();

else if (father->GetValue()->Top() > del\_elem) father->SetLeft(curr->GetLeft());

else father->SetRight(curr->GetLeft());

curr->SetLeft(nullptr);

curr = nullptr;

return;

}

if (!curr->LeftIns()) {

if (father == nullptr) this->Root = curr->GetRight();

else if (father->GetValue()->Top() > del\_elem) father->SetLeft(curr->GetRight());

else father->SetRight(curr->GetRight());

curr->SetRight(nullptr);

curr = nullptr;

return;

}

std::shared\_ptr<TTreeNode> currd = curr->GetRight();

std::shared\_ptr<TTreeNode> prev = curr;

while (currd->GetLeft() != nullptr) {

prev = currd;

currd = currd->GetLeft();

}

curr->SetValue(currd->GetValue());

if (prev == curr) curr->SetRight(currd->GetRight());

else prev->SetLeft(currd->GetRight());

currd->SetRight(nullptr);

currd = nullptr;

return;

}

friend std::ostream& operator<<(std::ostream& os, const TTree& obj) {

if (obj.Root == nullptr) {

std::cout << "Tree is empty" << std::endl;

return os;

}

os << \*obj.Root;

return os;

}

~TTree() {

this->Root = nullptr;

}

private:

std::shared\_ptr<TTreeNode> Root;

};

#endif

Листинг файла main.cpp

#include <iostream>

#include <cstdlib>

#include <memory>

#include <thread>

#include "TTreeNode.h"

#include "TTree.h"

#include "Figure.h"

#include "FourSquare.h"

#include "Rectangle.h"

#include "Trapeze.h"

void Help() {

std::cout << "To print tree enter: p" << std::endl;

std::cout << "To add figure to tree enter: a [name\_of\_figure] [side]" << std::endl;

std::cout << "To delete figure from tree enter: d [name\_of\_figure] [side]" << std::endl;

std::cout << "To exit programm enter: e" << std::endl;

std::cout << "To see this messege again enter: h\n" << std::endl;

std::cout << "Names of figures:\ns - square\nr - rectangle\nt - trapeze\n" << std::endl;

}

void ThrFunc(std::shared\_ptr<TTreeNode<Figure>> T);

void DFS(std::shared\_ptr<TTreeNode<Figure>> T);

void ThrFunc(std::shared\_ptr<TTreeNode<Figure>> T) {

DFS(T);

return;

}

void DFS(std::shared\_ptr<TTreeNode<Figure>> T) {

if (T == nullptr) return;

std::thread thr1(ThrFunc, T->GetLeft());

thr1.join();

T->GetFigure()->Print(std::cout);

std::thread thr2(ThrFunc, T->GetRight());

thr2.join();

return;

}

int main() {

TTree<Figure> T;

Help();

char ch;

std::shared\_ptr<Figure> ptr;

while (true) {

std::cin >> ch;

if (ch == 'e') {

break;

} else if (ch == 'a') {

std::cin >> ch;

if (ch == 's') ptr = std::shared\_ptr<Figure>(new FourSquare(std::cin));

else if (ch == 'r') ptr = std::shared\_ptr<Figure>(new Rectangle(std::cin));

else if (ch == 't') ptr = std::shared\_ptr<Figure>(new Trapeze(std::cin));

else {

std::cout << "Wrong figure" << std::endl;

continue;

}

T.add(ptr);

} else if (ch == 'd') {

std::cin >> ch;

if (ch == 's') ptr = std::shared\_ptr<Figure>(new FourSquare(std::cin));

else if (ch == 'r') ptr = std::shared\_ptr<Figure>(new Rectangle(std::cin));

else if (ch == 't') ptr = std::shared\_ptr<Figure>(new Trapeze(std::cin));

else {

std::cout << "Wrong figure" << std::endl;

continue;

}

T.del(ptr);

} else if (ch == 'p') {

std::cout << T << std::endl;

} else if (ch == 'h') {

Help();

} else if (ch == 'b') {

DFS(T.GetRoot());

} else {

continue;

}

}

return 0;

}

Выводы:

В ходе выполнения лабораторной̆ работы был получен навык работы с потоками в С++. Я научился создавать потоки и работать с объектами в этих потоках. Были спроектированы и запрограммированы на языке С++ две сортировки динамических структур содержащих объекты классов фигур: быстрая сортировка и многопоточная быстрая сортировка.

Лабораторная работа №9

Цели:

* 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 контейнером 1-го уровня:
  + Генерация фигур со случайным значением параметров;
  + Печать контейнера на экран;
  + Удаление элементов со значением площади меньше определенного числа;
* В контейнер второго уровня поместить цепочку команд.
* Реализовать цикл, который проходит по всем командам в контейнере второго уровня и выполняет их, применяя к контейнеру первого уровня.

Код:

Листинг файла Source.cpp

#include <cstdlib>

#include "Figure.h"

#include "TTree.h"

#include <future>

#include <functional>

#include <random>

#include <thread>

int main() {

TStack<Figure> tree\_figure;

typedef std::function<void (void) > command;

TTree < command> tree\_cmd;

command cmd\_insert = [&]() {

std::cout << "Command: Create figures" << std::endl;

std::default\_random\_engine generator;

std::uniform\_int\_distribution<int> distribution(1, 1000);

for (int i = 0; i < 10; i++) {

int side = distribution(generator);

tree\_figure.push(new FourSquare(side));

}

};

command cmd\_print = [&]() {

std::cout << "Command: Print tree" << std::endl;

std::cout << tree\_figure;

};

command cmd\_reverse = [&]() {

std::cout << "Command: Reverse tree" << std::endl;

TStack<Triangle> tree\_tmp;

while(!tree\_figure.empty()) tree\_tmp.push(tree\_figure.pop\_last());

while(!tree\_tmp.empty())tree\_figure.push(tree\_tmp.pop());

};

tree\_cmd.push(std::shared\_ptr<command> (&cmd\_print, [](command\*) {

}));

tree\_cmd.push(std::shared\_ptr<command> (&cmd\_reverse, [](command\*) {

}));

tree\_cmd.push(std::shared\_ptr<command> (&cmd\_print, [](command\*) {

}));

tree\_cmd.push(std::shared\_ptr<command> (&cmd\_insert, [](command\*) {

}));

while (!tree\_cmd.empty()) {

std::shared\_ptr<command> cmd = tree\_cmd.pop();

std::future<void> ft = std::async(\*cmd);

ft.get();

}

return 0;

}

Выводы:

В процессе выполнения данной̆ лабораторной̆ работы я познакомился с лямбда-выражениями. Лямбда-выражение используют для определения анонимного объекта-функции непосредственно в месте его вызова или передачи в функцию в качестве аргумента.