Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Отчёт по лабораторным работам по курсу «Объектно-ориентированное программирование»

Студент: Ю.О.Валентинова
Преподаватель: А.В.Поповкин
Группа: М8О-207Б
Вариант: 10
Дата:
Оценка:
Подпись:

Москва, 2017

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 расположены 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.

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ОПИСАНИЕ

Функция	Описание
Rectangle(size_t i, size_t j)	Конструктор с параметрами
void Print()	Печать
Rectangle(std::istream &is)	Конструктор из входного потока
double SquareF()	Функция нахождения площади
Rectangle()	Стандартный конструктор
Square()	Стандартный конструктор
Square(std::istream &is)	Конструктор из входного потока
Square(size_t i)	Конструктор с параметрами
Trapeze()	Стандартный конструктор
Trapeze(std::istream &is)	Конструктор из входного потока
Trapeze(long int i, long int j, long int k)	Конструктор с параметрами
bool IsErrorCin(std::istream &is)	Проверка на корректность ввода

КОНСОЛЬ

Main.h

```
#include <cstdlib>
#include <string>
#include "Trapeze.h"
#include "Rectangle.h"
#include "Square.h"
#include "ErrorCin.h"
void PrintAndSquare(Figure*);

int main(int argc, char** argv) {
	//setlocale(LC_ALL, "Russian");
	std::string name;
	std::cout << "Hi!" << std::endl;
	while (name != "Exit")
	{
		std::cout << "Which figure would you like to try?Enter
\"Trapezoid/Rectangle/Square\" or \"Exit\" for escape. " << std::endl;
		std::cin>>name;
		if (name == "Trapezoid" || name == "1") {
			Figure *ptr = new Trapeze(std::cin);
			PrintAndSquare(ptr);
		}
		else
			if (name == "Rectangle" || name == "2") {
				Figure *ptr = new Rectangle(std::cin);
				PrintAndSquare(ptr);
			}
			else
				if (name == "Square" || name == "3") {
					Figure *ptr = new Square(std::cin);
					PrintAndSquare(ptr);
				}
				else
					if(name!="Exit")
					std::cout << "Error. I don't know such
figure."<<std::endl;
	}
```

```
	system("pause");
	return 0;
}

void PrintAndSquare(Figure *ptr) {
	if (!ptr->error) {
		std::cout << "Your parameters are: ";
		ptr->Print();
		std::cout << "S=" << ptr->SquareF() << std::endl;
	}
	else {
		std::cout << "I can't solve it.Please, try again." << std::endl;
	}
	delete ptr;
}
```

Square.h

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

class Square : public Figure {
public:
	Square();
	Square(std::istream &is);
	Square(size_t i);

	double SquareF() override;
	void   Print() override;

	virtual ~Square();
private:
	size_t side_a;
};

#endif /* SQUARE_H */
```

Square.cpp

```
#include "Square.h"
#include"ErrorCin.h"
#include <iostream>
#include <cmath>

Square::Square() : Square(0) {
}

Square::Square(long int i) : side_a(i) {
	std::cout << "Square created: " << side_a << std::endl;
}

Square::Square(std::istream &is) {
	ErrorCin *ePtr = new ErrorCin;
	std::cout << "Side=";
	is >> side_a;
	if (side_a < 0) {
		error = true;
	}
```

```
	error=ePtr->IsErrorCin(is);
	delete(ePtr);
}

double Square::SquareF() {

	return side_a*side_a;
}

void Square::Print() {
	std::cout << "a=" << side_a << std::endl;

}

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

Figure.h

```
#ifndef FIGURE_H
#define	FIGURE_H

class Figure {
public:
	bool error = true;
	virtual double SquareF() = 0;
	virtual void   Print() = 0;
	virtual ~Figure() {};
};

#endif /* FIGURE_H */
```

Trapeze.h

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

class Trapeze : public Figure {
public:
	Trapeze();
	Trapeze(std::istream &is);
	Trapeze(long int i, long int j, long int k);

	double SquareF() override;
	void   Print() override;

	virtual ~Trapeze();
private:
	long int side_a;
	long int side_b;
	long int height;
};

#endif /* TRAPEZE_H */
```

Trapeze.cpp

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

Trapeze::Trapeze() : Trapeze(0, 0, 0) {
}

Trapeze::Trapeze(long int i, long int j, long int k) : side_a(i), side_b(j), height(k)
{
	std::cout << "Trapeze created: " << side_a << ", " << side_b << ", " << height
<< std::endl;
}

Trapeze::Trapeze(std::istream &is) {
	ErrorCin *ePtr = new ErrorCin;
	std::cout << "Side a and side b - two parallel sides." << std::endl;
	std::cout << "Side a=";
	is >> side_a;
	error=ePtr->IsErrorCin(is);
	std::cout << "Side b=";
	is >> side_b;
	error=ePtr->IsErrorCin(is);
	std::cout << "Height=";
	is >> height;
	error=ePtr->IsErrorCin(is);
	delete(ePtr);
}

double Trapeze::SquareF() {

	return height*(side_a+side_b)*0.5;

}

void Trapeze::Print() {
	std::cout << "a=" << side_a << ", b=" << side_b << ", height=" << height <<
std::endl;

}

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

ErrorCin.h

```
#ifndef ERRORCIN_H
#define	ERRORCIN_H
#include <iostream>

class ErrorCin {
public:
	bool error = false;
	bool IsErrorCin(std::istream &is) {
		if (is.fail()) { //устонавливает бит fail если is отличен от чисел
			error = true;
			is.clear();//очищает бит fail
			is.ignore(is.rdbuf()->in_avail());//очищение потока cin
			std::cout << "Error.Number is needed." << std::endl;
			//is.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
//-works
```

```
		}
		return error;
	}
	virtual ~ErrorCin() {};
};

#endif
```

Rectangle.h

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

class Rectangle : public Figure {
public:
	Rectangle();
	Rectangle(std::istream &is);
	Rectangle(size_t i, size_t j);

	double SquareF() override;
	void   Print() override;

	virtual ~Rectangle();
private:
	size_t side_a;
	size_t side_b;
};

#endif /* RECTANGLE_H */
```

Rectangle.cpp

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

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

Rectangle::Rectangle(size_t i, size_t j) : side_a(i), side_b(j) {
	std::cout << "Rectangle created: " << side_a << ", " << side_b << std::endl;
}

Rectangle::Rectangle(std::istream &is) {
	ErrorCin *ePtr = new ErrorCin;
	std::cout << "Side a=";
	is >> side_a;
	error=ePtr->IsErrorCin(is);
	std::cout << "Side b=";
	is >> side_b;
	error=ePtr->IsErrorCin(is);
	delete ePtr;
}

double Rectangle::SquareF() {

	return side_a*side_b;
```

```
}

void Rectangle::Print() {
	std::cout << "a=" << side_a << ", b=" << side_b << std::endl;

}

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

КОНСОЛЬ

Hi!
Which figure would you like to try?Enter "Trapezoid/Rectangle/Square" or "Exit" for escape.
Square
Side=2
Your parameters are: a=2
S=4
Square deleted
Which figure would you like to try?Enter "Trapezoid/Rectangle/Square" or "Exit" for escape.
Trapezoid
Side a and side b - two parallel sides.
Side a=3
Side b=4
Height=5
Your parameters are: a=3, b=4, height=5
S=17.5
Trapeze deleted
Which figure would you like to try?Enter "Trapezoid/Rectangle/Square" or "Exit" for escape.
cwdwe
Error. I don't know such figure.
Which figure would you like to try?Enter "Trapezoid/Rectangle/Square" or "Exit" for escape.
Trapezoid
Side a and side b - two parallel sides.
Side a=csdsd
Error.Number is needed.
Side b=
0
Height=0
I can't solve it.Please, try again.
Trapeze deleted
Which figure would you like to try?Enter "Trapezoid/Rectangle/Square" or "Exit" for escape.
Exit
Press any key to continue . . .

ВЫВОДЫ

В данной работе я познакомилась с объектно-ориентированной парадигмой программирования в языке C++. Изучила понятие класса и синтаксис объявления родительских-дочерних классов. Познакомилась с деструкторами, о существовании которых доселе не знала. Научилась использовать дружественные функции для доступа к приватным и защищённым полям и методам внутри различных классов.

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 расположены 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.

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ОПИСАНИЕ

Функция	Описание
TListItem(const Square& square)	Конструктор элемента списка
TListItem(const TListItem& orig)	Конструктор копирования элемента списка
TListItem* SetNext(TListItem* next)	Установить следующий элемент
TListItem* GetNext()	Получить следующий элемент
Square GetSquare() const	Найти площадь
virtual ~TListItem()	Деструктор
TList()	Конструктор списка
TList(const TList& orig)	Конструктор копирования списка
void addFirst(Square &&square)	Добавление элемента списка в начало
void addLast(Square &&square)	Добавление элемента списка в конец
void insert(Square &&square, Square &&squareNext)	Вставка элемента
bool empty()	Проверка на пустоту
Square getElement(int n)	Получение элемента
void delElement(Square &&square)	Удаление элемента
void eraseList()	Очистить список
virtual ~TList()	Деструктор

ФАЙЛЫ ПРОЕКТА

TList.cpp

```
#include "TList.h"

TList::TList() : first(nullptr) {
}

TList::TList(const TList& orig) {
	first = orig.first;
}

std::ostream& operator<<(std::ostream& os, const TList& list) {

	TListItem *item = list.first;

	while (item != nullptr)
	{
		os << *item;
		item = item->GetNext();
	}

	return os;
}
```

```
void TList::addFirst(Square &&square) {
	TListItem *other = new TListItem(square);
	other->SetNext(first);
	first = other;
}

void TList::addLast(Square &&square) {
	TListItem *other = new TListItem(square);
	TListItem *iter = this->first;
	if (first != nullptr) {
		while (iter->GetNext() != nullptr) {
			iter = iter->SetNext(iter->GetNext());
		}
		iter->SetNext(other);// little bit strange
		other->SetNext(nullptr);
	}
	else {
		first=other;
	}
}

void TList::insert(Square &&squareNext, Square &&square) {
	TListItem *other = new TListItem(square);
	TListItem *iter = this->first;
	if (iter->GetSquare() == squareNext) {
		other->SetNext(iter->GetNext());
		iter->SetNext(other);
		//other->SetNext(nullptr);
		std::cout << "Square is added." << std::endl;
	}
	else {
		while (!(iter->GetSquare() == squareNext) && (iter->GetNext() !=
nullptr)) {
			iter = iter->GetNext();
		}

		if ((iter->GetNext() == nullptr)&&(!(iter->GetSquare() == squareNext)))
{
			std::cout << "There is not such element in this list." <<
std::endl;
		}
		else {
			other->SetNext(iter->GetNext());
			iter->SetNext(other);
			std::cout << "Square is added." << std::endl;
		}
	}
}
bool TList::empty() {
	return first == nullptr;
}
Square TList::getElement(int n){
	TListItem* iter = this->first;
	for (int i = 1; i < n; i++) {
		iter = iter->GetNext();

	}
	return iter->GetSquare();
}
void TList::delElement(Square && square)
{
```

```
	TListItem* iter = this->first;
	if (iter != nullptr) {
		if (iter->GetSquare() == square) {
			first = nullptr;
			std::cout << "Square is deleted." << std::endl;
		}
		else {
			if (!(iter->GetNext() == nullptr)) {
				while (!(iter->GetNext() == nullptr) && !(iter->GetNext()-
>GetSquare() == square)) {
					iter = iter->GetNext();
				}
				if (!(iter->GetNext() == nullptr)) {
					iter->SetNext(iter->GetNext()->GetNext());
					std::cout << "Square is deleted." << std::endl;
				}
				else {
					std::cout << "There is no such element!" <<
std::endl;
				}
			}
			else {
				std::cout << "There is no such element!" << std::endl;
			}
		}
	}
}
void TList::eraseList() {
	first = nullptr;
}

TList::~TList() {
	std::cout << "List deleted!" << std::endl;
	delete first;
}
```

TList.h

```
#ifndef TLIST_H
#define	TLIST_H

#include "Square.h"
#include "TListItem.h"

class TList {
public:
	TList();
	TList(const TList& orig);

	void addFirst(Square &&square);
	void addLast(Square &&square);
	void insert(Square &&square, Square &&squareNext);
	bool empty();
	Square getElement(int n);
	void delElement(Square &&square);
	void eraseList();
	friend std::ostream& operator<<(std::ostream& os, const TList& stack);
	virtual ~TList();
private:

	TListItem *first;
};
```

```
#endif /* TLIST_H */
```

TListItem.cpp

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

TListItem::TListItem(const Square& square) {
	this->square = square;
	this->next = nullptr;
	//std::cout << "List item: created" << std::endl;
}

TListItem::TListItem(const TListItem& orig) {
	this->square = orig.square;
	this->next = orig.next;
	//std::cout << "List item: copied" << std::endl;
}

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

Square TListItem::GetSquare() const {
	return this->square;
}

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

TListItem::~TListItem() {
	std::cout << "List item: deleted"<< std::endl;
	delete next;

}

std::ostream& operator<<(std::ostream& os, const TListItem& obj) {
	os << "[" << obj.square << "]";//<< std::endl;
	return os;
}

bool operator==(TListItem & first, TListItem &last){
	return first.square == last.square;
}
```

TListItem.h

```
#ifndef TLISTITEM_H
#define	TLISTITEM_H

#include "Square.h"
class TListItem {
public:
	TListItem(const Square& square);
	TListItem(const TListItem& orig);//copy constr
	friend std::ostream& operator<<(std::ostream& os, const TListItem& obj);
	friend bool operator==(TListItem& first, TListItem& last);

	TListItem* SetNext(TListItem* next);
```

```
        TListItem* GetNext();
        Square GetSquare() const;

        virtual ~TListItem();
private:
        Square square;
        TListItem *next;
};

#endif /* TLISTITEM_H */
```

КОНСОЛЬ

Hello! That's my MENU:
1. Add new item in end of list.
2. Add new item in begin of list.
3. Delete item from list
4. Print list.
5. Insert in list
6. Erase list.
7. Print MENU.
0. Exit out program.

Input from 1 to 7 or 0 for actions.
1
Please, enter a side of the square:
3
Square is added.

Input from 1 to 7 or 0 for actions.
2
Please, enter a side of the square:
5
Square is added.

Input from 1 to 7 or 0 for actions.
4
[a=5][a=3]

Input from 1 to 7 or 0 for actions.
3
Please, enter a side of the square you want to delete:
5
Square is deleted.

Input from 1 to 7 or 0 for actions.
6
List is absolutely clean.

Input from 1 to 7 or 0 for actions.

4

_____
Input from 1 to 7 or 0 for actions.
0

ВЫВОДЫ

В данной работе я спроектировала свой АТД – список. Использование объектов намного упрощает работу с подобными типами данных. Кроме того, различные спецификаторы доступа делают код более читаемым, а сама ООП парадигма позволяет привязывать функции к определённым объектам, делая код более чистым. Фигуры я передавала «по значению», чтобы избежать чрезмерного копирования достаточно тяжелых объектов. Следует отметить, что стандартные контейнеры уже реализованы в мощнейшем инструменте программиста – STL.

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 (опередеяется
структурой контейнера).
• Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опередеяется
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.

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ОПИСАНИЕ

Функция	Описание
TListItem(const std::shared_ptr<Figure> &figure)	Конструктор элемента списка
std::shared_ptr<TListItem> SetNext(std::shared_ptr<TListItem> next)	Установить следующий элемент
std::shared_ptr<TListItem> GetNext()	Получить следующий элемент
std::shared_ptr<Figure> GetFigure()	Найти фигуру
virtual ~TListItem()	Деструктор
TList()	Конструктор списка
void addFirst(std::shared_ptr<Figure> &figure))	Добавление элемента списка в начало
void addLast(std::shared_ptr<Figure>& figure)	Добавление элемента списка в конец
bool empty()	Проверка на пустоту
Square getElement(int n)	Получение элемента
void delElement(Square &&square)	Удаление элемента
void eraseList()	Очистить список
virtual ~TList()	Деструктор

ФАЙЛЫ ПРОЕКТА

TList.cpp

```
#include "TList.h"

TList::TList() {
	first = nullptr;
}

std::ostream& operator<<(std::ostream& os, const TList& list) {

	std::shared_ptr<TListItem> item = list.first;
	int i = 1;
	while (item != nullptr)
	{
		std::cout << "[" << i << "]";
		item->GetFigure()->Print();
		item = item->GetNext();
		i++;
	}

	return os;
}

int TList::length() {
	int i = 0;
	std::shared_ptr<TListItem> item = this->first;
	while (item != nullptr)
	{
		item = item->GetNext();
		i++;
	}
	return i;
}
```

```
void TList::addFirst(std::shared_ptr<Figure> &figure) {
	std::shared_ptr<TListItem> other = std::make_shared<TListItem>(figure);

	other->SetNext(first);
	first = other;
}

void TList::insert(int index, std::shared_ptr<Figure> &figure) {
	std::shared_ptr<TListItem>iter = this->first;
	std::shared_ptr<TListItem> other = std::make_shared<TListItem>(figure);
	if (index == 1) {
		other->SetNext(iter);
		this->first = other;
	}
	else {
		if (index <= this->length()) {
			int i = 1;
			for (i = 1; i < index - 1; ++i) {
				iter = iter->GetNext();
			}
			other->SetNext(iter->GetNext());
			iter->SetNext(other);
		}
		else {
			std::cout << "error" << std::endl;
		}
	}
}

void TList::addLast(std::shared_ptr<Figure> &figure) {
	std::shared_ptr<TListItem> other = std::make_shared<TListItem>(figure);
	std::shared_ptr<TListItem> iter = this->first;
	if (first != nullptr) {
		while (iter->GetNext() != nullptr) {
			iter = iter->SetNext(iter->GetNext());
		}
		iter->SetNext(other);// little bit strange
		other->SetNext(nullptr);
	}
	else {
		first = other;
	}
}

bool TList::empty() {
	return first == nullptr;
}

void TList::delElement(int &index)
{
	std::shared_ptr<TListItem>iter = this->first;
	if (index <= this->length()) {
	if (index == 1) {
		this->first = iter->GetNext();
	}
	else {
		int i = 1;
		for (i = 1; i < index - 1; ++i) {
			iter = iter->GetNext();
		}
		iter->SetNext(iter->GetNext()->GetNext());
	}
```

```
	}
	else {
		std::cout << "error" << std::endl;
	}
}

void TList::eraseList() {
	first = nullptr;
}

TList::~TList() {
	std::cout << "List deleted!" << std::endl;
}
```

TList.h

```
#ifndef TLIST_H
#define	TLIST_H

#include "Square.h"
#include "Rectangle.h"
#include "Trapeze.h"
#include "TListItem.h"
#include <memory>

class TList {
public:
	TList();

	int length();

	void addFirst(std::shared_ptr<Figure> &figure);
	void insert(int index, std::shared_ptr<Figure>& figure);
	void addLast(std::shared_ptr<Figure>& figure);
	bool empty();
	void delElement(int & index);
	void eraseList();
	friend std::ostream& operator<<(std::ostream& os, const TList& stack);
	virtual ~TList();
private:

	std::shared_ptr<TListItem> first;
};

#endif /* TLIST_H */
```

TListItem.cpp

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

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

std::shared_ptr<TListItem>TListItem::SetNext(std::shared_ptr<TListItem> next) {
	std::shared_ptr<TListItem> old = this->next;
	this->next = next;
	return old;
}
```

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

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

TListItem::~TListItem() {
}

std::ostream& operator<<(std::ostream& os, const TListItem& obj) {
	os << "[" << obj.figure << "]";//<< std::endl;
	return os;
}
```

TListItem.h

```
#ifndef TLISTITEM_H
#define	TLISTITEM_H

#include "Square.h"
#include "Rectangle.h"
#include "Trapeze.h"
#include <memory>

class TListItem {
public:
	TListItem(const std::shared_ptr<Figure> &figure);
	//TListItem(const TListItem& orig);//copy constr
	friend std::ostream& operator<<(std::ostream& os, const TListItem& obj);

	std::shared_ptr<TListItem> SetNext(std::shared_ptr<TListItem> next);
	std::shared_ptr<TListItem> GetNext();
	std::shared_ptr<Figure> GetFigure();
	//Figure GetFigure() const;

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

#endif /* TLISTITEM_H */
```

КОНСОЛЬ

```
Hello! That's my MENU:
1. Add new item in begin of list.
2. Add new item in end of list.
3. Delete item from list
4. Print list.
5. Insert in list
6. Erase list.
7. Print MENU.
0. Exit out program.
_____
Input from 1 to 7 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
2
Side a=2
Side b=3
Figure is added.
_____
Input from 1 to 7 or 0 for actions.
2
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square).
1
Side a and side b - two parallel sides.
Side a=32
Side b=3
Height=4
Figure is added.
_____
Input from 1 to 7 or 0 for actions.
4
[1]a=2, b=3
[2]a=32, b=3, height=4

_____
Input from 1 to 7 or 0 for actions.
5
Please, enter a figure you want to add (1 - trapeze; 2- rectangle, 3 - square).
3
Side =2
Please, enter an index.
2
_____
Input from 1 to 7 or 0 for actions.
4
[1]a=2, b=3
[2]a=2
[3]a=32, b=3, height=4
```

ВЫВОДЫ

В данной работе я познакомилась с умными указателями в языке C++. Умные указатели считают ссылки на объект и когда количество ссылок становится равным нулю, то объект автоматически удаляется. Безусловно, это очень удобно, потому что риск утечек памяти становится минимальным. Есть и минусы, например, риск создания взаимоблокировок. В таком случае, следует использовать модификацию shared_ptr – weak_ptr.

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
(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).
Нельзя использовать:
• Стандартные контейнеры std.
Программа должна позволять:
• Вводить произвольное количество фигур и добавлять их в контейнер.
• Распечатывать содержимое контейнера.
• Удалять фигуры из контейнера.

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ОПИСАНИЕ

Функция	Описание
TListItem(const std::shared_ptr<T> &figure);	Конструктор элемента списка
std::shared_ptr<TListItem<T>> SetNext(std::shared_ptr<TListItem<T>> next)	Установить следующий элемент
std::shared_ptr<TListItem<T>> GetNext();	Получить следующий элемент
std::shared_ptr<T> GetFigure();	Найти фигуру
virtual ~TListItem()	Деструктор
TList()	Конструктор списка
void addFirst(std::shared_ptr<T> &figure);	Добавление элемента списка в начало
void addLast(std::shared_ptr<T> &figure);	Добавление элемента списка в конец
bool empty()	Проверка на пустоту
Square getElement(int n)	Получение элемента
void delElement(Square &&square)	Удаление элемента
void eraseList()	Очистить список
virtual ~TList()	Деструктор

ФАЙЛЫ ПРОЕКТА

TList.cpp

```
#include "TList.h"

TList::TList() {
	first = nullptr;
}

std::ostream& operator<<(std::ostream& os, const TList& list) {

	std::shared_ptr<TListItem> item = list.first;
	int i = 1;
	while (item != nullptr)
	{
		std::cout << "[" << i << "]";
		item->GetFigure()->Print();
		item = item->GetNext();
		i++;
	}

	return os;
}

int TList::length() {
	int i = 0;
	std::shared_ptr<TListItem> item = this->first;
	while (item != nullptr)
	{
		item = item->GetNext();
		i++;
	}
	return i;
}
void TList::addFirst(std::shared_ptr<Figure> &figure) {
```

```
	std::shared_ptr<TListItem> other = std::make_shared<TListItem>(figure);

	other->SetNext(first);
	first = other;
}

void TList::insert(int index, std::shared_ptr<Figure> &figure) {
	std::shared_ptr<TListItem>iter = this->first;
	std::shared_ptr<TListItem> other = std::make_shared<TListItem>(figure);
	if (index == 1) {
		other->SetNext(iter);
		this->first = other;
	}
	else {
		if (index <= this->length()) {
			int i = 1;
			for (i = 1; i < index - 1; ++i) {
				iter = iter->GetNext();
			}
			other->SetNext(iter->GetNext());
			iter->SetNext(other);
		}
		else {
			std::cout << "error" << std::endl;
		}
	}
}

void TList::addLast(std::shared_ptr<Figure> &figure) {
	std::shared_ptr<TListItem> other = std::make_shared<TListItem>(figure);
	std::shared_ptr<TListItem> iter = this->first;
	if (first != nullptr) {
		while (iter->GetNext() != nullptr) {
			iter = iter->SetNext(iter->GetNext());
		}
		iter->SetNext(other);// little bit strange
		other->SetNext(nullptr);
	}
	else {
		first = other;
	}
}

bool TList::empty() {
	return first == nullptr;
}

void TList::delElement(int &index)
{
	std::shared_ptr<TListItem>iter = this->first;
	if (index <= this->length()) {
	if (index == 1) {
		this->first = iter->GetNext();
	}
	else {
		int i = 1;
		for (i = 1; i < index - 1; ++i) {
			iter = iter->GetNext();
		}
		iter->SetNext(iter->GetNext()->GetNext());
	}
```

```
	}
	else {
		std::cout << "error" << std::endl;
	}
}

void TList::eraseList() {
	first = nullptr;
}

TList::~TList() {
	std::cout << "List deleted!" << std::endl;
}
```

TList.h

```
#ifndef TLIST_H
#define	TLIST_H

#include "Square.h"
#include "Rectangle.h"
#include "Trapeze.h"
#include "TListItem.h"
#include <memory>

class TList {
public:
	TList();

	int length();

	void addFirst(std::shared_ptr<Figure> &figure);
	void insert(int index, std::shared_ptr<Figure>& figure);
	void addLast(std::shared_ptr<Figure>& figure);
	bool empty();
	void delElement(int & index);
	void eraseList();
	friend std::ostream& operator<<(std::ostream& os, const TList& stack);
	virtual ~TList();
private:

	std::shared_ptr<TListItem> first;
};

#endif /* TLIST_H */
```

TListItem.cpp

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

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

std::shared_ptr<TListItem>TListItem::SetNext(std::shared_ptr<TListItem> next) {
	std::shared_ptr<TListItem> old = this->next;
	this->next = next;
	return old;
}
```

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

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

TListItem::~TListItem() {
}

std::ostream& operator<<(std::ostream& os, const TListItem& obj) {
	os << "[" << obj.figure << "]";//<< std::endl;
	return os;
}
```

TListItem.h

```
#ifndef TLISTITEM_H
#define	TLISTITEM_H

#include "Square.h"
#include "Rectangle.h"
#include "Trapeze.h"
#include <memory>

class TListItem {
public:
	TListItem(const std::shared_ptr<Figure> &figure);
	//TListItem(const TListItem& orig);//copy constr
	friend std::ostream& operator<<(std::ostream& os, const TListItem& obj);

	std::shared_ptr<TListItem> SetNext(std::shared_ptr<TListItem> next);
	std::shared_ptr<TListItem> GetNext();
	std::shared_ptr<Figure> GetFigure();
	//Figure GetFigure() const;

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

#endif /* TLISTITEM_H */
```

КОНСОЛЬ

Hello! That's my MENU:
1. Add new item in begin of list.
2. Add new item in end of list.
3. Delete item from list
4. Print list.
5. Insert in list
6. Erase list.
7. Print MENU.

0. Exit out program.

_____

Input from 1 to 7 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
2
Side a=2
Side b=3
Figure is added.

_____

Input from 1 to 7 or 0 for actions.
2
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square).
1
Side a and side b - two parallel sides.
Side a=32
Side b=3
Height=4
Figure is added.

_____

Input from 1 to 7 or 0 for actions.
4
[1]a=2, b=3
[2]a=32, b=3, height=4

_____

Input from 1 to 7 or 0 for actions.
5
Please, enter a figure you want to add (1 - trapeze; 2- rectangle, 3 - square).
3
Side =2
Please, enter an index.
2

_____

Input from 1 to 7 or 0 for actions.
4
[1]a=2, b=3
[2]a=2
[3]a=32, b=3, height=4

ВЫВОДЫ

В данной работе я познакомилась с шаблонами в языке C++. Шаблоны реализуют одну из трёх основных парадигм ООП – полиморфизм. Шаблоны позволяют принимать на вход функции или классу любой из встроенных типов данных, а также пользовательские типы. В зависимости от того, что нам нужно. Безусловно, шаблоны крайне важны. Они обеспечивают повторное использование программного кода, однако делают код менее читаемым.

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.

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ФАЙЛЫ ПРОЕКТА

TIterator.h

```
#ifndef TITERATOR_H
#define TITERATOR_H

#include <memory>
#include <iostream>

template <class N, class T>
class TIterator
{
public:
	TIterator(std::shared_ptr<N> n) {
		cur = n;
	}

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

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

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

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

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

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

private:
	std::shared_ptr<N> cur;
};

#endif
```

КОНСОЛЬ

Hello! That's my MENU:
1. Add new item in begin of list.
2. Add new item in end of list.
3. Delete item from list
4. Print list.
5. Insert in list

```
6. Erase list.
7. Print MENU.
8. Print list with iterator.
0. Exit out program.
_____
Input from 1 to 7 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
2
Side a=23
Side b=23
Figure is added.
_____
Input from 1 to 7 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
3
Side =23
Figure is added.
_____
Input from 1 to 7 or 0 for actions.
8
a=23
a=23, b=23
Input from 1 to 7 or 0 for actions.
0
Press any key to continue . . .
```

ВЫВОДЫ

В данной работе я познакомилась с итераторами в языке C++. Был разработан итератор для списка. Итераторы нужны для обеспечения доступа к элементам некоторого контейнера. Итераторы являются фундаментальным компонентом STL.

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.

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ОПИСАНИЕ

Функция	Описание
TAllocationBlock(int32_t size, int32_t count);	Конструктор класса
void *Allocate();	Выделение памяти
void Deallocate(void *ptr);	Освобождение памяти
bool Empty();	Проверка, пуст ли аллокатор
int32_t Size();	Получение количества выделенных блоков
virtual ~TAllocationBlock();	Деструктор класса
TTree();	Конструктор дерева
TTree(const TTree& orig);	Конструктор копирования дерева
void Insert(TNode*& node, void * link);	Вставка в дерево
void Pop();	Удаление минимального элемента
TNode* Root();	Нахождение значения корня
bool Empty() const;	Проверка на пустоту
TNode* Minimum() const;	Нахождение минимального элемента
void DistructTree(TNode* node);	Удаление дерева
virtual ~TTree();	Дествуктор

ФАЙЛЫ ПРОЕКТА

TAllocationBlock.h

```
#include <iostream>
#include <cstdlib>
#include "TTree.h"

typedef unsigned char Byte;

class TAllocationBlock
{
public:
	TAllocationBlock(int32_t size, int32_t count);
	void *Allocate();
	void Deallocate(void *ptr);
	bool Empty();
	int32_t Size();

	virtual ~TAllocationBlock();

private:
	Byte *_used_blocks;
	TTree _free_blocks;
};

#endif /* TALLOCATIONBLOCK_H */
```

TAllocationBlock.cpp

```
#include "TAllocationBlock.h"
```

```
#include <iostream>


TAllocationBlock::TAllocationBlock(int32_t size, int32_t count)
{
	_used_blocks = (Byte *)malloc(size * count);

	for (int32_t i = 0; i < count; ++i) {
		void *ptr = (void *)malloc(sizeof(void *));
		ptr = _used_blocks + i * size;
		_free_blocks.Insert(_free_blocks.root, ptr);
	}
}

void *TAllocationBlock::Allocate()
{
	if (!_free_blocks.Empty()) {
		void *res = _free_blocks.Minimum();
		_free_blocks.Pop();
		return res;
	}
	else {
		throw std::bad_alloc();
	}
}

void TAllocationBlock::Deallocate(void *ptr)
{
	_free_blocks.Insert(_free_blocks.root, ptr);
}

bool TAllocationBlock::Empty()
{
	return _free_blocks.Empty();
}

int32_t TAllocationBlock::Size()
{
	return _free_blocks.size;
}

TAllocationBlock::~TAllocationBlock()
{
	while (!_free_blocks.Empty()) {
		_free_blocks.Pop();
	}
	free(_used_blocks);
}
```

КОНСОЛЬ

Hello! That's my MENU:
1. Add new item in begin of list.
2. Add new item in end of list.
3. Delete item from list
4. Print list.
5. Insert in list
6. Erase list.
7. Print MENU.

```
8. Print list with iterator.
0. Exit out program.
_____
Input from 1 to 7 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
2
Side a=23
Side b=2
Figure is added.
_____
Input from 1 to 7 or 0 for actions.
3
Please, enter an index of figure you want to delete
1
_____
Input from 1 to 7 or 0 for actions.
2
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square).
3
Side =34
Figure is added.
_____
Input from 1 to 7 or 0 for actions.
4
[1]a=34

_____
Input from 1 to 7 or 0 for actions.
8
a=34
Input from 1 to 7 or 0 for actions.
2
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square).
3
Side =56
Figure is added.
_____
Input from 1 to 7 or 0 for actions.
0
Press any key to continue . . .
```

ВЫВОДЫ

В данной работе я познакомилась с аллокаторами памяти в языке C++. Я написала некое подобие аллокатора для реализованного ранее списка. Для хранения свободных блоков пришлось реализовать N-дерево, что не так практично, но, как оказалось, вполне возможно сделать. Полученный аллокатор я использовала, переопределив new и delete в классе списка.

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:
o По типу (например, все квадраты).
o По площади (например, все объекты с площадью меньше чем заданная).

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ОПИСАНИЕ

Функция	Описание
tree.inorder()	Печать
tree.removeByType(index);	Удаление по типу
tree.removeLesser(sqr);	Удаление по площади

ФАЙЛЫ ПРОЕКТА

Main.cpp

```
#include "TList.h"
#include <iostream>
#include "Square.h"
#include "Trapeze.h"
#include "Rectangle.h"
#include "TTree.h"

void menu(void) {
  std::cout << "1) Add item" << std::endl;
  std::cout << "2) Print" << std::endl;
  std::cout << "3) Delete by criteria" << std::endl;
  std::cout << "4) Exit" << std::endl;
}

int main(void) {
  TTree<TList<Figure>, std::shared_ptr<Figure> > tree;
  int opt, index;
  Square tmp1;
  Trap tmp2;
  Pryam tmp3;

  do {
    menu();
    std::cin >> opt;
    switch(opt) {
    case 1:{
      std::cout << "Enter 1 for square, 2 for trapeze, 3 for rectangle" <<
std::endl;
      std::cin >> index;
      if (index == 1) {
        std::cout << "Enter value" << std::endl;
        std::cin >> tmp1;
        tree.insert(std::make_shared<Square>(tmp1));
        std::cout << "Item was added" << std::endl;
        break;
      } else if (index == 2) {
        std::cout << "Enter value" << std::endl;
        tmp2.setParams(std::cin);
        tree.insert(std::make_shared<Trap>(tmp2));
        std::cout << "Item was added" << std::endl;
        break;
      } else if (index == 3) {
        std::cout << "Enter value" << std::endl;
        tmp3.setParams(std::cin);
        tree.insert(std::make_shared<Pryam>(tmp3));
        std::cout << "Item was added" << std::endl;
        break;
```

```
      } else {
        std::cout << "derp" << std::endl;
        break;
      }
    }
    case 2:
      tree.inorder();
      break;
    case 3:{
      std::cout << "Enter criteria" << std::endl;
      std::cout << "1) by type\n2) lesser than square\n";
      std::cin >> index;
      if (index == 1) {
        std::cout << "1) square\n2) trapeze\n3) rectangle\n";
        std::cout << "Enter type" << std::endl;
        std::cin >> index;
        tree.removeByType(index);
      } else if (index == 2) {
        double sqr = 0.0;
        std::cout << "Enter square" << std::endl;
        std::cin >> sqr;
        tree.removeLesser(sqr);
      } else {
        break;
      }
      break;
    }
    }
  } while(opt != 4);

    std::cout << "Bye!" << std::endl;
  return 0;
}
```

TList.hpp

```
#ifdef TLIST_H

template <typename Q, typename O> TTree<Q, O>::TTree() {
  root = std::make_shared<Node>(Node());
  root->son = std::make_shared<Node>(Node());
}

template <typename Q, typename O> TTree<Q, O>::Node::Node() {
  son = sibling = nullptr;
  itemsInNode = 0;
}

template <typename Q, typename O> TTree<Q, O>::Node::Node(const O& item) {
  data.PushFront(item);
  itemsInNode = 1;
}

template <typename Q, typename O> void TTree<Q,
O>::recRemByType(std::shared_ptr<Node>& node, const int& type) {
  if (node->itemsInNode) {

    for (int i = 0; i < 5; i++) {
      auto iter = node->data.begin();
```

```
      for (int k = 0; k < node->data.GetLength(); k++) {
        if (iter->type() == type) {
          node->data.Pop(k + 1);
          node->itemsInNode--;
          break;
        }
        ++iter;
      }
    }

    if (node->sibling) {
      recRemByType(node->sibling, type);
    }
    if (node->son) {
      recRemByType(node->son, type);
    }

  }
}

template <typename Q, typename O> void TTree<Q, O>::removeByType(const int&
type) {
  recRemByType(root->son, type);
}

template <typename Q, typename O> void TTree<Q,
O>::recInsert(std::shared_ptr<Node>& node, const O& item) {
  if (node->itemsInNode < 5) {
    node->data.PushFront(item);
    node->itemsInNode++;
  } else {
    auto sibl = node;

    for (int i = 0; i < 3; i++) {
      if (!sibl->sibling) {
        sibl->sibling = std::make_shared<Node>(Node(item));
        return;
      }

      if (sibl->sibling->itemsInNode < 5) {
        recInsert(sibl->sibling, item);
        return;
      }

      sibl = sibl->sibling;
    }

    if (node->son) {
      recInsert(node->son, item);
    } else {
      node->son = std::make_shared<Node>(Node(item));
    }
  }
}

template <typename Q, typename O> void TTree<Q, O>::insert(const O& item) {
  recInsert(root->son, item);
}

template <typename Q, typename O> void TTree<Q, O>::recInorder(const
std::shared_ptr<Node>& node) {
```

```
  if (node->itemsInNode) {
    node->data.sort();
    for (const auto& i: node->data) {
      i->Print();
    }
    std::cout << "\n";

    if (node->sibling) {
      recInorder(node->sibling);
    }
    if (node->son) {
      recInorder(node->son);
    }
  }
}

template <typename Q, typename O> void TTree<Q, O>::inorder() {
  if (root->son->son || root->son->sibling) {
    clear(root->son, root);
  }
  recInorder(root->son);
}

template <typename Q, typename O> void TTree<Q,
O>::recRemLesser(std::shared_ptr<Node>& node, const double& sqr) {
  if (node->itemsInNode) {
    for (int i = 0; i < 5; i++) {
      auto iter = node->data.begin();

      for (int k = 0; k < node->data.GetLength(); k++) {
        if (iter->getSquare() < sqr) {
          node->data.Pop(k + 1);
          node->itemsInNode--;
          break;
        }
        ++iter;
      }
    }

    if (node->sibling) {
      recRemLesser(node->sibling, sqr);
    }
    if (node->son) {
      recRemLesser(node->son, sqr);
    }

  }
}

template <typename Q, typename O> void TTree<Q, O>::removeLesser(const
double& sqr) {
  recRemLesser(root->son, sqr);
}

template <typename Q, typename O> void TTree<Q,
O>::clear(std::shared_ptr<Node>& node, std::shared_ptr<Node>& parent) {
  if (node) {
    if (!node->itemsInNode) {
      auto orphan = node;
      auto orphanPar = parent;
      if (node->sibling) {
```

```
        orphan = node->sibling;
        orphanPar = node;
      } else if (node->son) {
        orphan = node->sibling;
        orphanPar = node;
      }

      while (orphan->sibling || orphan->son) {
        orphanPar = orphan;
        if (orphan->sibling) {
          orphan = orphan->sibling;
        } else if (orphan->son) {
          orphan = orphan->son;
        }
      }

      if (orphanPar->sibling == orphan) {
        std::swap(node->data, orphan->data);
        node->itemsInNode = orphan->itemsInNode;
        orphanPar->sibling = nullptr;
      } else if (orphanPar->son == orphan) {
        std::swap(node->data, orphan->data);
        node->itemsInNode = orphan->itemsInNode;
        orphanPar->son = nullptr;
      }

    }
  }

  if (node) {
    if (node->son) {
      clear(node->son, node);
    }
    if (node->sibling) {
      clear(node->sibling, node);
    }
  }
}

#endif
```

КОНСОЛЬ

```
ju@vav:~/inf/lab7$ make
g++ -pedantic -std=c++14 main.cpp Square.cpp Trapeze.cpp Rectangle.cpp TAllocator.cpp -
o lab7 -lpthread
ju@vav:~/inf/lab7$ ./lab7
1) Add item
2) Print
3) Delete by criteria
4) Exit
1
Enter 1 for square, 2 for trapeze, 3 for rectangle
1
Enter value
23
Item was added
```

1) Add item
2) Print
3) Delete by criteria
4) Exit
1
Enter 1 for square, 2 for trapeze, 3 for rectangle
2
Enter value
2
2
2
2
Item was added
1) Add item
2) Print
3) Delete by criteria
4) Exit
3
Enter criteria
1) by type
2) lesser than square
1
1) square
2) trapeze
3) rectangle
Enter type
2
1) Add item
2) Print
3) Delete by criteria
4) Exit
2
Type of figure is square
a = 23

1) Add item
2) Print
3) Delete by criteria
4) Exit
1
Enter 1 for square, 2 for trapeze, 3 for rectangle
1
Enter value
4
Item was added
1) Add item
2) Print
3) Delete by criteria
4) Exit
3

```
Enter criteria
1) by type
2) lesser than square
2
Enter square
17
1) Add item
2) Print
3) Delete by criteria
4) Exit
2
Type of figure is square
a = 23

1) Add item
2) Print
3) Delete by criteria
4) Exit
4
```

ВЫВОДЫ

В данной работе я углубила свои знания в шаблонном проектировании классов в языке C++. Был усовершенствован контейнер второго уровня. Теперь удалять элементы из контейнера можно по площади, либо по типу фигуры.

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

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ОПИСАНИЕ

Функция	Описание
void TList<T>::Sort()	Простая сортировка
void TList<T>::ParSort()	Параллельная сортировка
std::shared_ptr<TListItem<T>> TList<T>::PParSort(std::shared_ptr<TListItem<T>> &first)	Дополнительная функция для параллельной сортировки
std::shared_ptr<TListItem<T>> TList<T>::Partition(std::shared_ptr<TListItem<T>> &first)	Разбиение на части
std::shared_ptr<TListItem<T>> TList<T>::PSort(std::shared_ptr<TListItem<T>> &first)	Дополнительная функция для параллельной сортировки

ФАЙЛЫ ПРОЕКТА

Main.cpp

```
#include <cstdlib>
#include <iostream>
#include "Figure.h"
#include "Square.h"
#include "Rectangle.h"
#include "Trapeze.h"
#include "TListItem.h"
#include "TList.h"
#include "TTree.h"
#include "Rectangle.h"




int main(int argc, char** argv) {
	TList<Figure> list;
	int menuNum = 7;
	int size;
	int figure;
	std::shared_ptr<Figure> sptr, sptrTemp;
	sptr = std::make_shared<Square>();
	std::cout << "Hello! That's my MENU:" << std::endl;
	while (menuNum != 0) {
		if ((menuNum >= 0) || (menuNum <= 7))
		{
			switch (menuNum)
			{
			case 0:
				std::cout << "Bye!" << std::endl;
				break;
			case 1:
				std::cout << "Please, enter a figure (1 - trapeze; 2-
rectangle, 3 - square) " << std::endl;
				std::cin >> figure;
				if (figure == 1) {
					sptr = std::make_shared<Trapeze>(std::cin);
					list.addFirst(sptr);
					std::cout << "Figure is added." << std::endl;
				}
```

```
                        else if(figure == 2){
                                sptr = std::make_shared<Rectangle>(std::cin);
                                list.addFirst(sptr);
                                std::cout << "Figure is added." << std::endl;
                        }
                        else if (figure == 3) {
                                sptr = std::make_shared<Square>(std::cin);
                                list.addFirst(sptr);
                                std::cout << "Figure is added." << std::endl;
                        }
                        else {
                                std::cout << "I can't make such figure." <<
std::endl;
                        }
                        std::cout << "_____________________________________" <<
std::endl;
                        break;
                case 2:
                        std::cout << "Please, enter a figure (1 - trapeze; 2-
rectangle, 3 - square). " << std::endl;
                        std::cin >> figure;
                        if (figure == 1) {
                                sptr = std::make_shared<Trapeze>(std::cin);
                                list.addLast(sptr);
                                std::cout << "Figure is added." << std::endl;
                        }
                        else if (figure == 2) {
                                sptr = std::make_shared<Rectangle>(std::cin);
                                list.addLast(sptr);
                                std::cout << "Figure is added." << std::endl;
                        }
                        else if (figure == 3) {
                                sptr = std::make_shared<Square>(std::cin);
                                list.addLast(sptr);
                                std::cout << "Figure is added." << std::endl;
                        }
                        else {
                                std::cout << "I can't make such figure." <<
std::endl;
                        }
                        std::cout << "_____________________________________" <<
std::endl;
                        break;
                case 3:
                        //sptr = std::make_shared<Trapeze>(std::cin);
                        //list.addFirst(sptr);
                        //std::cout << "_____________________________________" <<
std::endl;
                        //break;
                        std::cout << "Please, enter an index of figure you want to
delete " << std::endl;
                        std::cin >> size;
                        if (size < 0) {
                                std::cout << "I don't know such index." <<
std::endl;
                                break;
                        }
                        list.delElement(size);
                        std::cout << "_____________________________________" <<
std::endl;
                        break;
                case 4:
```

```
                        std::cout << list << std::endl;
                        std::cout << "_____________________________________" <<
std::endl;
                        break;
                case 5:
                        std::cout << "Please, enter a figure you want to add (1 -
trapeze; 2- rectangle, 3 - square)." << std::endl;
                        std::cin >> figure;
                        if (figure == 1) {
                                sptr = std::make_shared<Trapeze>(std::cin);
                        }
                        else if (figure == 2) {
                                sptr = std::make_shared<Rectangle>(std::cin);
                        }
                        else if (figure == 3) {
                                sptr = std::make_shared<Square>(std::cin);
                        }
                        else {
                                std::cout << "I can't find such figure." <<
std::endl;
                        }
                        std::cout << "Please, enter an index." << std::endl;
                        std::cin >> size;
                        list.insert(size,sptr);
                        std::cout << "_____________________________________" <<
std::endl;
                        break;

                case 6:
                        list.eraseList();
                        std::cout << "List is absolutely clean." << std::endl;
                        std::cout << "_____________________________________" <<
std::endl;
                        break;
                case 7:
                        std::cout << "1. Add new item in begin of list." <<
std::endl;
                        std::cout << "2. Add new item in end of list." <<
std::endl;
                        std::cout << "3. Delete item from list" << std::endl;
                        std::cout << "4. Print list." << std::endl;
                        std::cout << "5. Insert in list" << std::endl;
                        std::cout << "6. Erase list." << std::endl;
                        std::cout << "7. Print MENU." << std::endl;
                        std::cout << "8. Print list with iterator." << std::endl;
                        std::cout << "9. Sort list." << std::endl;
                        std::cout << "0. Exit out program." << std::endl;
                        std::cout << "_____________________________________" <<
std::endl;
                        break;
                case 8:
                        for (auto i : list) {
                                i->Print();
                        }
                        break;
                case 9:
                        std::cout << "1 to regular sort, 2 to parallel" <<
std::endl;
                        std::cin >> menuNum;
                        int temp=1;
                        if (menuNum == 1) {
                                list.addFirst(sptr);
```

```
                                list.Sort();
                                list.delElement(temp);
                        }
                        else if (menuNum == 2) {
                                list.addFirst(sptr);
                                list.ParSort();
                                list.delElement(temp);
                        }
                        else {
                                std::cout << "Unknown command" << std::endl;
                                break;
                        }
                        std::cout << list << std::endl;
                        break;
                }

                //default:
                //std::cout<<"I hardly understand you!"<<std::endl;
        }
        else {
                std::cout << "I hardly understand you!" << std::endl;
        }
        std::cout << "Input from 1 to 7 or 0 for actions." << std::endl;
        std::cin >> menuNum;

    }


    //std::cout << list<<std::endl;
    system("pause");
    return 0;
}
```

TList.cpp

```
#include "TList.h"
#include "TIterator.h"

template <class T>
TList<T>::TList() {
    first = nullptr;
}

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

    std::shared_ptr<TListItem<T>> item = list.first;
    int i = 1;
    while (item != nullptr)
    {
        std::cout << "[" << i << "]";
        item->GetFigure()->Print();
        item = item->GetNext();
        i++;
    }

    return os;
}

template <class T>
int TList<T>::length() {
    int i = 0;
    std::shared_ptr<TListItem<T>> item = this->first;
```

```
	while (item != nullptr)
	{
		item = item->GetNext();
		i++;
	}
	return i;
}

template <class T>
void TList<T>::addFirst(std::shared_ptr<T> &figure) {
	std::shared_ptr<TListItem<T>> other = std::make_shared<TListItem<T>>(figure);

	other->SetNext(first);
	first = other;
}

template <class T>
void TList<T>::insert(int index, std::shared_ptr<T> &figure) {
	std::shared_ptr<TListItem<T>>iter = this->first;
	std::shared_ptr<TListItem<T>> other = std::make_shared<TListItem<T>>(figure);
	//int i = 0;
	if (index == 1) {
		other->SetNext(iter);
		this->first = other;
	}
	else {
		if (index <= this->length()) {
			int i = 1;
			for (i = 1; i < index - 1; ++i) {
				iter = iter->GetNext();
			}
			other->SetNext(iter->GetNext());
			iter->SetNext(other);
		}
		else {
			std::cout << "error" << std::endl;
		}
	}
}

template <class T>
void TList<T>::addLast(std::shared_ptr<T> &figure) {
	std::shared_ptr<TListItem<T>> other = std::make_shared<TListItem<T>>(figure);
	std::shared_ptr<TListItem<T>> iter = this->first;
	if (first != nullptr) {
		while (iter->GetNext() != nullptr) {
			iter = iter->SetNext(iter->GetNext());
		}
		iter->SetNext(other);// little bit strange
		other->SetNext(nullptr);
	}
	else {
		first = other;
	}
}

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

template <class T>
```

```
void TList<T>::delElement(int &index)
{
	std::shared_ptr<TListItem<T>>iter = this->first;
	//std::shared_ptr<TListItem> other = std::make_shared<TListItem>(figure);
	//int i = 0;
	if (index <= this->length()) {
	if (index == 1) {
		this->first = iter->GetNext();
	}
	else {
		int i = 1;
		for (i = 1; i < index - 1; ++i) {
			iter = iter->GetNext();
		}
		iter->SetNext(iter->GetNext()->GetNext());
	}

	}
	else {
		std::cout << "error" << std::endl;
	}
}

template <class T>
void TList<T>::eraseList() {
	first = nullptr;
}

template <class T>
TList<T>::~TList() {
	std::cout << "List deleted!" << std::endl;
	//delete first;
}
template <class T>
TIterator<TListItem<T>, T> TList<T>::begin()
{
	return TIterator<TListItem<T>, T>(first);
}

template <class T>
TIterator<TListItem<T>, T> TList<T>::end()
{
	return TIterator<TListItem<T>, T>(nullptr);
}

template <class T>
std::shared_ptr<TListItem<T>> TList<T>::PSort(std::shared_ptr<TListItem<T>> &first)
{
	if (first == nullptr || first->GetNext() == nullptr) {
		return first;
	}

	std::shared_ptr<TListItem<T>> partitionedEl = Partition(first);
	std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
	std::shared_ptr<TListItem<T>> rightPartition = first;

	partitionedEl->SetNext(nullptr);

	if (leftPartition == nullptr) {
		leftPartition = first;
		rightPartition = first->GetNext();
		first->SetNext(nullptr);
```

```
	}

	rightPartition = PSort(rightPartition);
	leftPartition = PSort(leftPartition);
	std::shared_ptr<TListItem<T>> iter = leftPartition;
	while (iter->GetNext() != nullptr) {
		iter = iter->GetNext();
	}

	iter->SetNext(rightPartition);

	return leftPartition;
}

template <class T>
std::shared_ptr<TListItem<T>> TList<T>::Partition(std::shared_ptr<TListItem<T>>
&first)
{
	std::lock_guard<std::mutex> lock(mutex);
	if (first->GetNext()->GetNext() == nullptr) {
		if (first->GetNext()->GetFigure()->SquareF() > first->GetFigure()-
>SquareF()) {
			return first->GetNext();
		}
		else {
			return first;
		}
	}
	else {
		std::shared_ptr<TListItem<T>> i = first->GetNext();
		std::shared_ptr<TListItem<T>> pivot = first;
		std::shared_ptr<TListItem<T>> lastElSwapped = (pivot->GetNext()-
>GetFigure()->SquareF() >= pivot->GetFigure()->SquareF()) ? pivot->GetNext() : pivot;

		while ((i != nullptr) && (i->GetNext() != nullptr)) {
			if (i->GetNext()->GetFigure()->SquareF() >= pivot->GetFigure()-
>SquareF()) {
				if (i->GetNext() == lastElSwapped->GetNext()) {
					lastElSwapped = lastElSwapped->GetNext();
				}
				else {
					std::shared_ptr<TListItem<T>> tmp = lastElSwapped-
>GetNext();
					lastElSwapped->SetNext(i->GetNext());
					i->SetNext(i->GetNext()->GetNext());
					lastElSwapped = lastElSwapped->GetNext();
					lastElSwapped->SetNext(tmp);
				}
			}
			i = i->GetNext();
		}
		return lastElSwapped;
	}
}

template <class T>
void TList<T>::Sort()
{
	if (first == nullptr)
		return;
	std::shared_ptr<TListItem<T>> tmp = first->GetNext();
	first->SetNext(PSort(tmp));
```

```
}

template <class T>
void TList<T>::ParSort()
{
	if (first == nullptr)
		return;
	std::shared_ptr<TListItem<T>> tmp = first->GetNext();
	first->SetNext(PParSort(tmp));
}

template <class T>
std::shared_ptr<TListItem<T>> TList<T>::PParSort(std::shared_ptr<TListItem<T>> &first)
{
	if (first == nullptr || first->GetNext() == nullptr) {
		return first;
	}

	std::shared_ptr<TListItem<T>> partitionedEl = Partition(first);
	std::shared_ptr<TListItem<T>> leftPartition = partitionedEl->GetNext();
	std::shared_ptr<TListItem<T>> rightPartition = first;

	partitionedEl->SetNext(nullptr);

	if (leftPartition == nullptr) {
		leftPartition = first;
		rightPartition = first->GetNext();
		first->SetNext(nullptr);
	}

	std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&
)>
		task1(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
	std::packaged_task<std::shared_ptr<TListItem<T>>(std::shared_ptr<TListItem<T>>&
)>
		task2(std::bind(&TList<T>::PParSort, this, std::placeholders::_1));
	auto rightPartitionHandle = task1.get_future();
	auto leftPartitionHandle = task2.get_future();

	std::thread(std::move(task1), std::ref(rightPartition)).join();
	rightPartition = rightPartitionHandle.get();
	std::thread(std::move(task2), std::ref(leftPartition)).join();
	leftPartition = leftPartitionHandle.get();
	std::shared_ptr<TListItem<T>> iter = leftPartition;
	while (iter->GetNext() != nullptr) {
		iter = iter->GetNext();
	}

	iter->SetNext(rightPartition);
	return leftPartition;
}
#include "Figure.h"
template class TList<Figure>;
template std::ostream& operator<<(std::ostream &out, const TList<Figure> &figure);
```

КОНСОЛЬ

```
Hello! That's my MENU:
1. Add new item in begin of list.
2. Add new item in end of list.
```

3. Delete item from list
4. Print list.
5. Insert in list
6. Erase list.
7. Print MENU.
8. Print list with iterator.
9. Sort list.
0. Exit out program.

Input from 1 to 9 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
3
Side =1
Figure is added.

Input from 1 to 9 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
3
Side =2
Figure is added.

Input from 1 to 9 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
3
Side =1
Figure is added.

Input from 1 to 9 or 0 for actions.
4
[1]a=1
[2]a=2
[3]a=1

Input from 1 to 9 or 0 for actions.
9
1 to regular sort, 2 to parallel
1
[1]a=1
[2]a=1
[3]a=2

Input from 1 to 9 or 0 for actions.
1
Please, enter a figure (1 - trapeze; 2- rectangle, 3 - square)
3
Side =34

```
Figure is added.
_____________________________________________
Input from 1 to 9 or 0 for actions.
4
[1]a=34
[2]a=1
[3]a=1
[4]a=2

_____________________________________________
Input from 1 to 9 or 0 for actions.
9
1 to regular sort, 2 to parallel
2
[1]a=1
[2]a=1
[3]a=2
[4]a=34

Input from 1 to 9 or 0 for actions.
0
Press any key to continue . . .
```

ВЫВОДЫ

В данной работе я познакомилась с параллельным программированием в языке C++. Научилась использовать базовые примитивы синхронизации, такие как mutex. Кроме того, я узнала, как в C++ можно создавать потоки и работать с ними. Среда разработки Microsoft Visual Studio отлично подходит для отладки многопоточных приложений, так как не только показывает количество работающих потоков, но и позволяет пошагово отлаживать каждый поток.

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:
o Генерация фигур со случайным значением параметров;
o Печать контейнера на экран;
o 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.

ВАРИАНТ

Контейнер 1 уровня: связный список
Контейнер 2 уровня: N-дерево
Фигуры: квадрат, прямоугольник, трапеция.

ОПИСАНИЕ

Описание классов и структур, определённых ранее, осталось без изменений

ФАЙЛЫ ПРОЕКТА

Main.cpp

```
#include "TList.h"
#include <iostream>
#include "Square.h"
#include "Trapeze.h"
#include "Rectangle.h"
#include "TTree.h"
#include <random>

int main(void) {
  TList<Figure> list;
  typedef std::function<void(void)> command;
  TTree<std::shared_ptr<command> > tree(4);
  command cmdInsert = [&]() {
    std::cout << "Command: Insert" << std::endl;
    std::default_random_engine generator;
    std::uniform_int_distribution<int> distribution(1, 10);
    for (int i = 0; i < 10; i++) {
      int side = distribution(generator);
      if ((side % 2) == 0) {
        list.PushFront(std::make_shared<Trap>(Trap(side, side, side,
side)));
      } else if((side % 3) == 0) {
        list.PushFront(std::make_shared<Pryam>(Pryam(side, side + 1)));
      } else {
        list.PushFront(std::make_shared<Square>(Square(side)));
      }
    }
  };
  command cmdPrint = [&]() {
    std::cout << "Command: Print" << std::endl;
    for (const auto& i : list) {
      i->Print();
    }
  };
  command cmdRemove = [&]() {
    std::cout << "Command: Remove" << std::endl;
    std::default_random_engine generator;
    std::uniform_real_distribution<double> distribution(1.0, 150.0);
    double sqr = distribution(generator);
    std::cout << "Lesser than " << sqr << std::endl;
    for (int i = 0; i < 10; i++) {
      auto iter = list.begin();

      for (int k = 0; k < list.GetLength(); k++) {
        if (iter->getSquare() < sqr) {
          list.Pop(k + 1);
          break;
        }
        ++iter;
      }
    }
  };
```

```
  tree.insert(std::shared_ptr<command>(&cmdInsert, [](command*){});
  tree.insert(std::shared_ptr<command>(&cmdPrint, [](command*){});
  tree.insert(std::shared_ptr<command>(&cmdRemove, [](command*){});
  tree.insert(std::shared_ptr<command>(&cmdPrint, [](command*){});
  tree.inorder();

  return 0;
}
```

КОНСОЛЬ

```
ju@vav:~/inf/lab9$ make
g++ -pedantic -std=c++14 main.cpp Square.cpp Trapeze.cpp Rectangle.cpp TAllocator.cpp -
o lab9 -lpthread
ju@vav:~/inf/lab9$ ./lab9
Command: Insert
Command: Print
Type of figure is trapeze
a = 10
b = 10
c = 10
d = 10
Type of figure is square
a = 7
Type of figure is square
a = 7
Type of figure is square
a = 1
The type of figure is rectangle
a = 3
b = 4
Type of figure is trapeze
a = 6
b = 6
c = 6
d = 6
Type of figure is square
a = 5
Type of figure is trapeze
a = 8
b = 8
c = 8
d = 8
Type of figure is trapeze
a = 2
b = 2
c = 2
d = 2
Type of figure is square
a = 1
```

```
Command: Remove
Lesser than 20.5991
Command: Print
Type of figure is trapeze
a = 10
b = 10
c = 10
d = 10
Type of figure is square
a = 7
Type of figure is square
a = 7
Type of figure is trapeze
a = 6
b = 6
c = 6
d = 6
Type of figure is square
a = 5
Type of figure is trapeze
a = 8
b = 8
c = 8
d = 8
Type of figure is trapeze
a = 2
b = 2
c = 2
d = 2
```

ВЫВОДЫ

В данной работе я познакомилась с лямбда-выражениями в языке C++. Лямбда-выражения присущи в основном функциональным языкам программирования. Однако и в C++, и в C# использование лямбда-функций уже не вызывает удивления. Они упрощают написание кода и делают его более компактным. Однако начинающим программистам читать код с лямбда-выражениями намного сложнее.

Git Hub

Ссылки на все лабораторные работы по курсу:

1. https://github.com/juliavav/MAI_OOP_01
2. https://github.com/juliavav/MAI_OOP_02
3. https://github.com/juliavav/MAI_OOP_03
4. https://github.com/juliavav/MAI_OOP_04
5. https://github.com/juliavav/MAI_OOP_05
6. https://github.com/juliavav/MAI_OOP_06
7. https://github.com/juliavav/MAI_OOP_07
8. https://github.com/juliavav/MAI_OOP_08
9. https://github.com/juliavav/MAI_OOP_09