

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы
по курсу
Объектно-ориентированное программирование

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №1

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

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.
- Должны иметь общий виртуальный метод расчета площади фигуры – Square.
- Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin.
- Должны быть расположены в отдельных 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.

Теория

Классы и объекты в C++ являются основными концепциями объектно-ориентированного программирования.

Классы в C++ — это абстракция описывающая методы, свойства, присущие сразу группе объектов.

Объекты — конкретное представление абстракции, имеющее свои свойства. Созданные объекты на основе одного класса называются экземплярами этого класса. Эти объекты могут иметь различное поведение, свойства, но все равно будут являться объектами одного класса. В ООП существует три основных принципа построения классов:

Инкапсуляция — это свойство, позволяющее объединить в классе и данные, и методы, работающие с ними и скрыть детали реализации от пользователя.

Наследование — это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя присваиваются классу-потомку.

Полиморфизм — свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Перегрузка — это возможность поддерживать несколько функций с одним названием, но разными сигнатурами вызова.

Дружественная функция — это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях `private` или `protected`

Листинг:

- Figure.h

```

#ifndef FIGURE_H
#define FIGURE_H

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

#endif /* FIGURE_H */

```

- Pentagon.h

```

#ifndef LAB_1_PENTAGON_H
#define LAB_1_PENTAGON_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Pentagon : public Figure {
public:
    Pentagon ();
    Pentagon (std::istream &is);
    Pentagon (size_t side, size_t height);
    Pentagon (const Pentagon& orig);
    double Square() override;
    void Print() override;
    virtual ~Pentagon();

private:
    size_t side;
    size_t apothem;
};

#endif //LAB_1_PENTAGON_H

```

- Pentagon.cpp

```

#include "Pentagon.h"
#include <iostream>
//apothem

Pentagon::Pentagon() : Pentagon(0,0) {
}

Pentagon::Pentagon(std::istream &is) {
    std::cout << "Enter side and apothem:"<<std::endl;
    is >> side >> apothem;
}

Pentagon::Pentagon(const Pentagon &orig) {
    side = orig.side;
    apothem = orig.apothem;
}

Pentagon::Pentagon(size_t side, size_t apothem): side(side), apothem(apothem) {
    std::cout << "Rhombus created: " << side << ", " << apothem << std::endl;
}

```

```

}

void Pentagon::Print() {
    std::cout << "Pentagon:" <<std::endl;
    std::cout <<"side = " << side << ", apothem = " << apothem << std::endl;
}

double Pentagon::Square() {
    return (side * apothem / 2) * 5 ;
}

Pentagon::~~Pentagon() {
    std::cout << "Pentagon delete" << std::endl;
}

```

- Rhombus.h

```

#ifndef LAB_1_RHOMBUS_H
#define LAB_1_RHOMBUS_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Rhombus : public Figure {
public:

    Rhombus ();
    Rhombus (std::istream &is);
    Rhombus (size_t side, size_t height);
    Rhombus (const Rhombus& orig);
    double Square() override;
    void Print() override;
    virtual ~Rhombus();

private:
    size_t side;
    size_t height;

};

#endif //LAB_1_RHOMBUS_H

```

- Rhombus.cpp

```

#include "Rhombus.h"
#include <iostream>

Rhombus::Rhombus() : Rhombus(0,0) {

}

Rhombus::Rhombus(std::istream &is) {
    std::cout << "Enter side and height:"<<std::endl;
    is >> side >> height;
}

Rhombus::Rhombus(const Rhombus &orig) {
    side = orig.side;
    height = orig.height;
}

```

```

Rhombus::Rhombus(size_t side, size_t height): side(side), height(height) {
    std::cout << "Rhombus created: " << side << ", " << height << std::endl;
}

void Rhombus::Print() {
    std::cout << "Rhombus:" <<std::endl;
    std::cout <<"side = " << side << ", height = " << height << std::endl;
}

double Rhombus::Square() {
    return side * height;
}

Rhombus::~Rhombus() {
    std::cout << "Rhombus delete" << std::endl;
}

```

- Trapeze.h

```

#ifndef LAB_1_TRAPEZE_H
#define LAB_1_TRAPEZE_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Trapeze : public Figure {
public:
    Trapeze ();
    Trapeze (std::istream &is);
    Trapeze (size_t base_a, size_t base_b, size_t height);
    Trapeze (const Trapeze& orig);
    double Square() override;
    void Print() override;
    virtual ~Trapeze();

private:
    size_t base_a;
    size_t base_b;
    size_t height;

};

#endif //LAB_1_TRAPEZE_H

```

- Trapeze.cpp

```

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

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

Trapeze::Trapeze(std::istream &is) {
    std::cout << "Enter first base, second base and height:"<<std::endl;
    is >> base_a >> base_b >> height;
}

Trapeze::Trapeze(const Trapeze &orig) {

```

```

        base_a = orig.base_a;
        base_b = orig.base_b;
        height = orig.height;
    }

Trapeze::Trapeze(size_t base_a, size_t base_b, size_t height): base_a(base_a),
base_b(base_b), height(height) {
    std::cout << "Trapeze created: " << base_a << ", " << base_b << ", " <<
height << std::endl;
}

void Trapeze::Print() {
    std::cout << "Trapeze:" <<std::endl;
    std::cout <<"first base = " << base_a << ", second base = " << base_b << ",
height = " << height << std::endl;
}

double Trapeze::Square() {
    return ((base_a + base_b) / 2) * height;
}

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

```

- main.cpp

```

#include <cstdlib>
#include <iostream>
#include "Trapeze.h"
#include "Rhombus.h"
#include "Pentagon.h"

int main(int argc, char** argv) {
    char option = '0';
    Figure *ptr;

    while(true) {
        std::cout << std::endl << "Menu: " << std::endl;
        std::cout << "1) Trapeze" << std::endl;
        std::cout << "2) Rhombus" << std::endl;
        std::cout << "3) Pentagon" << std::endl;
        std::cout << "4) Exit" << std::endl;
        std::cout << "Enter the option:" << std::endl;
        std::cin >> option;
        if(option == '4') {
            break;
        }
        switch (option) {
            case '1':{
                ptr = new Trapeze(std::cin);
                break;
            }
            case '2':{
                ptr = new Rhombus(std::cin);
                break;
            }
            case '3':{
                ptr = new Pentagon(std::cin);
                break;
            }
            default:{
                std::cout << "Unknown option" << std::endl;;
            }
        }
    }
}

```

```
        break;
    }

    ptr->Print();
    std::cout << "Square:" << std::endl << ptr->Square() << std::endl;
    delete ptr;
}

return 0;
}
```

Выводы: в данной лабораторной работе я получил навыки программирования классов на языке C++, познакомился с перегрузкой операторов и дружественными функциями.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_1

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №2

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

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 (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 отдельных 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.

Теория

Динамические структуры данных – это структуры данных, память под которые выделяется и освобождается по мере необходимости.

Динамические структуры данных в процессе существования в памяти могут изменять

не только число составляющих их элементов, но и характер связей между элементами. При этом не учитывается изменение содержимого самих элементов данных. Такая особенность динамических структур, как непостоянство их размера и характера отношений между элементами, приводит к тому, что на этапе создания машинного кода программа-компилятор не может выделить для всей структуры в целом участок памяти фиксированного размера, а также не может сопоставить с отдельными компонентами структуры конкретные адреса. Для решения проблемы адресации динамических структур данных используется метод, называемый *динамическим распределением памяти*, то есть память под отдельные элементы выделяется в момент, когда они "начинают существовать" в процессе выполнения программы, а не во время компиляции. Компилятор в этом случае выделяет фиксированный объем памяти для хранения адреса динамически размещаемого элемента, а не самого элемента.

Динамическая структура данных характеризуется тем что:

- она не имеет имени; (чаще всего к ней обращаются через указатель на адрес)
- ей выделяется память в процессе выполнения программы;
- количество элементов структуры может не фиксироваться;
- размерность структуры может меняться в процессе выполнения программы;
- в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

При **передаче по значению** содержимое аргумента копируется в формальный параметр подпрограммы. Изменения, сделанные в параметре, не влияют на значение переменной, используемой при вызове.

Листинг

- Trapeze.h

```
#ifndef LAB_1_TRAPEZE_H
#define LAB_1_TRAPEZE_H
#include <cstdlib>
#include <iostream>
#include "Figure.h"

class Trapeze : public Figure {
public:

    Trapeze ();
    Trapeze (std::istream &is);
    Trapeze (size_t base_a, size_t base_b, size_t height);
    Trapeze (const Trapeze& orig);
    double Square() override;
    void Print() override;
    virtual ~Trapeze();

private:
    size_t base_a;
    size_t base_b;
    size_t height;

};

#endif //LAB_1_TRAPEZE_H
```

- Trapeze.cpp

```

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

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

Trapeze::Trapeze(std::istream &is) {
    std::cout << "Enter first base, second base and height:"<<std::endl;
    is >> base_a >> base_b >> height;
}

Trapeze::Trapeze(const Trapeze &orig) {
    base_a = orig.base_a;
    base_b = orig.base_b;
    height = orig.height;
}

Trapeze::Trapeze(size_t base_a, size_t base_b, size_t height): base_a(base_a),
base_b(base_b), height(height) {
    std::cout << "Trapeze created: " << base_a << ", " << base_b << ", " <<
height << std::endl;
}

void Trapeze::Print() {
    std::cout << "Trapeze:" <<std::endl;
    std::cout <<"first base = " << base_a << ", second base = " << base_b << ",
height = " << height << std::endl;
}

double Trapeze::Square() {
    return ((base_a + base_b) / 2) * height;
}

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

```

- TStack.h

```

#ifndef LAB_2_TSTACK_H
#define LAB_2_TSTACK_H

#include "TStackItem.h"

class TStack {
public:
    TStack();
    //copy constructor
    TStack(const TStack& orig);

    //for rvalue
    void push(Trapeze &&trap);
    //for lvalue
    void push(Trapeze &trap);
    size_t getSize();
    bool empty();
    Trapeze pop();

    friend std::ostream& operator<<(std::ostream &os, const TStack& stack);

    ~TStack();
}

```

```

private:
    TStackItem* top;
    size_t size;
};

#endif //LAB_2_TSTACK_H

• TStack.cpp
#include "TStack.h"

TStack::TStack() : top(nullptr), size(0) {
}

TStack::TStack(const TStack &orig) {
    TStackItem *item = orig.top;
    size = orig.size;
    while(item != nullptr) {
        item = item->GetPrev();
        this->push(item->GetTrapeze());
    }
}

size_t TStack::getSize() {
    return this->size;
}

void TStack::push(Trapeze &&trap) {
    TStackItem* newItem = new TStackItem(trap);
    newItem->SetPrev(top);
    top = newItem;
    size++;
}

void TStack::push(Trapeze &trap) {
    TStackItem* newItem = new TStackItem(trap);
    newItem->SetPrev(top);
    top = newItem;
    size++;
}

bool TStack::empty() {
    return top==nullptr;
}

Trapeze TStack::pop() {
    Trapeze result;
    if (top != nullptr) {
        TStackItem *old_top = top;
        top = top->GetPrev();
        result = old_top->GetTrapeze();
        old_top->SetPrev(nullptr);
        delete old_top;
        size--;
    }
    else {
        std::cout << "Stack isn't have elements" << std::endl;
    }

    return result;
}

```

```

}

TStack::~~TStack() {
    while(!this->empty()) {
        this->pop();
    }
}

std::ostream& operator<<(std::ostream &os, const TStack& stack) {
    TStackItem *item = stack.top();
    os << "Stack: " << std::endl;

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

• TstackItem.h
#ifndef LAB_2_TSTACKITEM_H
#define LAB_2_TSTACKITEM_H

#include "Trapeze.h"

class TStackItem {
public:
    //type conversion
    TStackItem(const Trapeze& trap);
    //copy construct
    TStackItem(const TStackItem& orig);

    friend std::ostream& operator<<(std::ostream& os, const TStackItem& item);

    void SetPrev(TStackItem* prev);
    TStackItem* GetPrev();
    Trapeze GetTrapeze() const;

    virtual ~TStackItem();
private:
    Trapeze trapeze;
    TStackItem* prev;
};

#endif //LAB_2_TSTACKITEM_H

• TStackItem.cpp

#include "TStackItem.h"

TStackItem::TStackItem(const Trapeze& trap) {
    trapeze = trap;
    prev = nullptr;
    //std::cout << "Stack item: created" << std::endl;
}

TStackItem::TStackItem(const TStackItem &orig) {
    trapeze = orig.trapeze;
    prev = orig.prev;
    //std::cout << "Stack item: copied" << std::endl;
}

```

```

}

void TStackItem::SetPrev(TStackItem *prev) {
    if(this != nullptr)
        this->prev = prev;
}

TStackItem* TStackItem::GetPrev() {
    return this->prev;
}

Trapeze TStackItem::GetTrapeze() const {
    return this->trapeze;
}

TStackItem::~TStackItem() {
    //std::cout << "Stack item: deleted" << std::endl;
    delete prev;
}

std::ostream& operator<<(std::ostream& os, const TStackItem& item) {
    std::cout << item.trapeze;
    return os;
}

```

- main.cpp

```

#include <cstdlib>
#include <iostream>
#include "TStack.h"

```

```

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

    TStack stack;
    unsigned int action;

    while (true) {
        std::cout << "Menu:" << std::endl;
        std::cout << "1) Push figure" << std::endl;
        std::cout << "2) Pop figure" << std::endl;
        std::cout << "3) Print" << std::endl;
        std::cout << "4) Print size" << std::endl;
        std::cout << "0) Quit" << std::endl;
        std::cin >> action;

        if (action == 0) {
            break;
        }

        switch (action) {
            case 1: {
                Trapeze t;
                std::cin >> t;
                stack.push(t);
                break;
            }
            case 2: {
                stack.pop();
                break;
            }

            case 3: {
                std::cout<<stack;

```

```
        break;
    }

    case 4: {
        size_t size = stack.getSize();
        std::cout << "size: " << size << std::endl;
        break;
    }
    default:
        std::cout << "Invalid action" << std::endl;
        break;
}

return 0;
}
```

Выводы: в данной лабораторной работе я закрепил навыки программирования классов на языке C++, изучил и применил динамические структуры.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_2

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №3

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

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 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 отдельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).
- Программа должна позволять:
- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Теория

Smart pointer — это объект, работать с которым можно как с обычным указателем, но при этом, в отличие от последнего, он предоставляет дополнительный функционал (например, автоматическое освобождение закрепленной за указателем области памяти).

Умные указатели призваны для борьбы с утечками памяти, которые сложно избежать в больших проектах. Они особенно удобны в местах, где возникают исключения, так как при последних происходит процесс раскрутки стека и уничтожаются локальные объекты. В случае обычного указателя — уничтожится переменная-указатель, при этом ресурс останется не освобожденным. В случае умного указателя — вызовется деструктор, который и освободит выделенный ресурс.

В новом стандарте появились следующие умные указатели: `unique_ptr`, `shared_ptr` и `weak_ptr`. Все они объявлены в заголовочном файле `<memory>`.

unique_ptr

Этот указатель пришел на смену старому и проблематичному `auto_ptr`. Основная проблема последнего заключается в правах владения. Объект этого класса теряет права владения ресурсом при копировании (присваивании, использовании в конструкторе копий, передаче в функцию по значению).

shared_ptr

Это самый популярный и самый широкоиспользуемый умный указатель. Он начал своё развитие как часть библиотеки `boost`. Данный указатель был столь успешным, что его включили в C++ Technical Report 1 и он был доступен в пространстве имен `tr1` — `std::tr1::shared_ptr<>`.

В отличие от рассмотренных выше указателей, `shared_ptr` реализует подсчет

ссылок на ресурс. Ресурс освободится тогда, когда счетчик ссылок на него будет равен 0. Как видно, система реализует одно из основных правил сборщика мусора.

weak_ptr

Этот указатель также, как и `shared_ptr` начал свое рождение в проекте boost, затем был включен в C++ Technical Report 1 и, наконец, пришел в новый стандарт. Данный класс позволяет избежать циклической зависимости, которая может образоваться при использовании `shared_ptr`.

Листинг

- **TStack.h**

```
#ifndef LAB_2_TSTACK_H
#define LAB_2_TSTACK_H

#include "TStackItem.h"

class TStack {
public:
    TStack();
    //copy constructor
    TStack(const TStack& orig);

    //for rvalue
    void push(std::shared_ptr<Figure> &&trap);
    //for lvalue
    void push(std::shared_ptr<Figure> &trap);
    size_t getSize();
    bool empty();
    std::shared_ptr<Figure> pop();

    friend std::ostream& operator<<(std::ostream &os, const TStack& stack);

    ~TStack();

private:
    std::shared_ptr<TStackItem> top;
    size_t size;
};

#endif //LAB_2_TSTACK_H
```

- **TStack.cpp**

```
#include "TStack.h"

TStack::TStack() : top(nullptr), size(0) {
}

TStack::TStack(const TStack& orig) {
    std::shared_ptr<TStackItem> item = orig.top;
    size = orig.size;
    while(item != nullptr) {
        item = item->GetPrev();
        this->push(item->GetFigure());
    }
}

size_t TStack::getSize() {
```

```

        return this->size;
    }

    void TStack::push(std::shared_ptr<Figure> &&trap) {
        std::shared_ptr<TStackItem> newItem(new TStackItem(trap));
        newItem->SetPrev(top);
        top = newItem;
        size++;
    }

    void TStack::push(std::shared_ptr<Figure> &trap) {
        std::shared_ptr<TStackItem> newItem(new TStackItem(trap));
        newItem->SetPrev(top);
        top = newItem;
        size++;
    }

    bool TStack::empty() {
        return top==nullptr;
    }

    std::shared_ptr<Figure> TStack::pop() {
        std::shared_ptr<Figure> result;
        if (top != nullptr) {
            std::shared_ptr<TStackItem>old_top = top;
            top = top->GetPrev();
            result = old_top->GetFigure();
            old_top->SetPrev(nullptr);
            size--;
        }
        else {
            std::cout << "Stack isn't have elements" << std::endl;
        }

        return result;
    }

    TStack::~TStack() {
        while(!this->empty()) {
            this->pop();
        }
    }

    std::ostream& operator<<(std::ostream &os, const TStack& stack) {
        std::shared_ptr<TStackItem>item = stack.top;
        os << "Stack: " << std::endl<< std::endl;

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

```

- TStackItem.h

```

#ifndef LAB_2_TSTACKITEM_H
#define LAB_2_TSTACKITEM_H

#include "Trapeze.h"
#include "Pentagon.h"
#include "Rhombus.h"

```

```

class TStackItem {
public:
    //type conversion
    TStackItem(const std::shared_ptr<Figure> &trap);
    //copy construct
    TStackItem(const TStackItem& orig);

    friend std::ostream& operator<<(std::ostream& os, const TStackItem& item);

    void SetPrev(std::shared_ptr<TStackItem> prev);
    std::shared_ptr<TStackItem> GetPrev();
    std::shared_ptr<Figure> GetFigure() const;

    virtual ~TStackItem();
private:
    std::shared_ptr<Figure> figure;
    std::shared_ptr<TStackItem> prev;
};

```

```

#endif //LAB_2_TSTACKITEM_H

```

- TStackItem.cpp

```

#include "TStackItem.h"

```

```

TStackItem::TStackItem(const std::shared_ptr<Figure> &trap) {
    figure = trap;
    prev = nullptr;
    //std::cout << "Stack item: created" << std::endl;
}

TStackItem::TStackItem(const TStackItem& orig) {
    figure = orig.figure;
    prev = orig.prev;
    //std::cout << "Stack item: copied" << std::endl;
}

void TStackItem::SetPrev(std::shared_ptr<TStackItem> prev) {
    if(this != nullptr)
        this->prev = prev;
}

std::shared_ptr<TStackItem> TStackItem::GetPrev() {
    return this->prev;
}

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

TStackItem::~~TStackItem() {
    //std::cout << "Stack item: deleted" << std::endl;
}

std::ostream& operator<<(std::ostream& os, const TStackItem& item) {
    std::cout << *item.figure;
    return os;
}

```

- main.c

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

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

    TStack stack;
    unsigned int action;

    while (true) {
        std::cout << "Menu:" << std::endl;
        std::cout << "1) Push figure" << std::endl;
        std::cout << "2) Pop figure" << std::endl;
        std::cout << "3) Print" << std::endl;
        std::cout << "4) Print size" << std::endl;
        std::cout << "0) Quit" << std::endl;
        std::cin >> action;

        if (action == 0) {
            break;
        }

        switch (action) {
            case 1: {
                char figure_type;
                std::shared_ptr<Figure> t;
                std::cout
                    << "Enter figure type: t - trapeze, p - pentagon, r -
rhombus: ";
                std::cin >> figure_type;
                switch (figure_type) {
                    case 't': {
                        t = std::make_shared<Trapeze>(std::cin);
                        stack.push(t);
                        break;
                    }
                    case 'p': {
                        t = std::make_shared<Pentagon>(std::cin);
                        stack.push(t);
                        break;
                    }
                    case 'r': {
                        t = std::make_shared<Rhombus>(std::cin);
                        stack.push(t);
                        break;
                    }
                    default: {
                        std::cout << "Try more..." << std::endl;
                        break;
                    }
                }
            }
            break;
        }
        case 2: {
            stack.pop();
            break;
        }
    }
}
```

```

        case 3: {
            std::cout<<stack;
            break;
        }

        case 4: {
            size_t size = stack.getSize();
            std::cout << "size: " << size << std::endl;
            break;
        }
        default:
            std::cout << "Invalid action" << std::endl;
            break;
    }
}

return 0;
}

```

Выводы: в данной лабораторной работе я закрепил навыки программирования классов на языке C++, изучил и применил умные указатели. Реализовал контейнер хранящий три различные фигуры по ссылке.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_3

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №4

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

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 содержать 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 (определяется структурой контейнера).
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера (определяется 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 расположены в отдельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Теория

Шаблоны — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

В C++ возможно создание шаблонов функций и классов.

Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой тип или значение одного из допустимых типов (целое число, enum, указатель на любой объект с глобально доступным именем, ссылка).

Любой шаблон начинается со слова `template`, будь то шаблон функции или шаблон класса. После ключевого слова `template` идут угловые скобки — `< >`, в которых перечисляется список параметров шаблона. Каждому параметру должно предшествовать зарезервированное слово `class` или `typename`. Отсутствие этих ключевых слов будет расцениваться компилятором как синтаксическая ошибка.

Ключевое слово `typename` говорит о том, что в шаблоне будет использоваться встроенный тип данных, такой как: `int`, `double`, `float`, `char` и т. д. А ключевое слово `class` сообщает компилятору, что в шаблоне функции в качестве параметра будут использоваться пользовательские типы данных, то есть классы. Но не в коем случае не путайте параметр шаблона и шаблон класса.

Листинг

- `TStack.h`

```
#ifndef LAB_2_TSTACK_H
#define LAB_2_TSTACK_H
```

```

#include "TStackItem.h"

template <class T>
class TStack {
public:
    TStack();
    //copy constructor
    TStack(const TStack<T>& orig);

    //for rvalue
    void push(T &&trap);
    //for lvalue
    void push(T &trap);
    size_t getSize();
    bool empty();
    T pop();

    template <class A>
    friend std::ostream& operator<<(std::ostream &os, const TStack<A>& stack);

    ~TStack();

private:
    std::shared_ptr<TStackItem<T>> top;
    size_t size;
};

#endif //LAB_2_TSTACK_H

```

- TStack.cpp

```

#include "TStack.h"

template <class T>
TStack<T>::TStack() : top(nullptr), size(0) {
}

template <class T>
TStack<T>::TStack(const TStack<T>& orig) {
    std::shared_ptr<TStackItem<T>> item = orig.top;
    size = orig.size;
    while(item != nullptr) {
        item = item->GetPrev();
        this->push(item->GetFigure());
    }
}

template <class T>
size_t TStack<T>::getSize() {
    return this->size;
}

template <class T>
void TStack<T>::push(T &&trap) {
    std::shared_ptr<TStackItem<T>> newItem(new TStackItem<T>(trap));
    newItem->SetPrev(top);
    top = newItem;
}

```

```

    size++;
}

template <class T>
void TStack<T>::push(T &trap) {
    std::shared_ptr<TStackItem<T>> newItem(new TStackItem<T>(trap));
    newItem->SetPrev(top);
    top = newItem;
    size++;
}

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

template <class T>
T TStack<T>::pop() {
    T result;
    if (top != nullptr) {
        std::shared_ptr<TStackItem<T>>old_top = top;
        top = top->GetPrev();
        result = old_top->GetFigure();
        old_top->SetPrev(nullptr);
        size--;
    }
    else {
        std::cout << "Stack isn't have elements" << std::endl;
    }

    return result;
}

template <class T>
TStack<T>::~~TStack() {
    while(!this->empty()) {
        this->pop();
    }
}

template <class T>
std::ostream& operator<<(std::ostream &os, const TStack<T>& stack) {
    std::shared_ptr<TStackItem<T>>item = stack.top;
    os << "Stack: " << std::endl<< std::endl;

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

```

```
template class TStack<std::shared_ptr<Figure>>;
template std::ostream& operator<<(std::ostream &os, const TStack<std::shared_ptr<Figure>>&
stack);
```

- TStackItem.h

```
#ifndef LAB_2_TSTACKITEM_H
```

```
#define LAB_2_TSTACKITEM_H
```

```
#include "../figure/Trapeze.h"
```

```
#include "../figure/Pentagon.h"
```

```
#include "../figure/Rhombus.h"
```

```
template <class T>
```

```
class TStackItem {
```

```
public:
```

```
    //type conversion
```

```
    TStackItem(const T &fig);
```

```
    //copy construct
```

```
    TStackItem(const TStackItem& orig);
```

```
    template <class A>
```

```
    friend std::ostream& operator<<(std::ostream& os, const TStackItem<A>&
item);
```

```
    void SetPrev(std::shared_ptr<TStackItem<T>> prev);
```

```
    std::shared_ptr<TStackItem<T>> GetPrev();
```

```
    T GetFigure() const;
```

```
    virtual ~TStackItem();
```

```
private:
```

```
    T figure;
```

```
    std::shared_ptr<TStackItem<T>> prev;
```

```
};
```

```
#endif //LAB_2_TSTACKITEM_H
```

- TStackItem.cpp

```
#include "TStackItem.h"
```

```
template <class T>
TStackItem<T>::TStackItem(const T &fig) {
    figure = fig;
    prev = nullptr;
    //std::cout << "Stack item: created" << std::endl;
}
```

```
template <class T>
TStackItem<T>::TStackItem(const TStackItem& orig) {
    figure = orig.figure;
    prev = orig.prev;
    //std::cout << "Stack item: copied" << std::endl;
}
```

```
template <class T>
void TStackItem<T>::SetPrev(std::shared_ptr<TStackItem<T>> prev) {
    if(this != nullptr)
        this->prev = prev;
}
```

```
template <class T>
std::shared_ptr<TStackItem<T>> TStackItem<T>::GetPrev() {
    return this->prev;
}
```

```
template <class T>
T TStackItem<T>::GetFigure() const{
    return this->figure;
}
```

```

template <class T>
TStackItem<T>::~~TStackItem() {
    //std::cout << "Stack item: deleted" << std::endl;

}

template <class T>
std::ostream& operator<<(std::ostream& os, const TStackItem<T>& item) {
    std::cout << *item.figure;
    return os;
}

template class TStackItem<std::shared_ptr<Figure>>;
template std::ostream& operator<<(std::ostream &os, const
TStackItem<std::shared_ptr<Figure>>& stack);

```

Выводы: в данной лабораторной работе я получил навыки построения шаблонов динамических структур данных. Реализовал шаблонный контейнер.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_4

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №5

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

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;`

Теория

Итератор (от англ. *iterator* — перечислитель) — интерфейс, предоставляющий доступ к элементам коллекции (массива или контейнера) и навигацию по ним.

Язык C++ широко использует итераторы в STL, поддерживающей несколько различных типов итераторов, включая 'однонаправленные итераторы', 'двунаправленные итераторы' и 'итераторы произвольного доступа'. Все стандартные шаблоны типов контейнеров реализуют разнообразный, но постоянный набор типов итераторов. Синтаксис стандартных итераторов сделан похожим на обычные указатели языка Си, где операторы `*` и `->` используются для указания элемента, на который указывает итератор, а такие арифметические операторы указателя, как `++`, используются для перехода итератора к следующему элементу.

Итераторы обычно используются парами, один из которых используется для указания текущей итерации, а второй служит для обозначения конца коллекции. Итераторы создаются при помощи соответствующих классов контейнеров, используя такие стандартные методы как `begin()` и `end()`. Функция `begin()` возвращает указатель на первый элемент, а `end()` — на воображаемый несуществующий элемент, следующий за последним.

Листинг

- TIterator.h

```
#ifndef LAB_4_TITERATOR_H
#define LAB_4_TITERATOR_H

#include <memory>
#include <iostream>

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

    TIterator(std::shared_ptr<node> n)    {

        node_ptr = n;
    }

    T operator * () {
        return node_ptr->GetFigure();
    }
};
```



```

    }

    T operator -> (){
        return node_ptr->GetFigure();
    }

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

    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<node> node_ptr;
};

```

```

#endif //LAB_4_TITERATOR_H

```

• TStack.h

```

#ifndef LAB_2_TSTACK_H
#define LAB_2_TSTACK_H

#include "TStackItem.h"
#include "TIterator.h"

template <class T>
class TStack {
public:
    TStack();
    //copy constructor
    TStack(const TStack<T>& orig);

    //for rvalue
    void push(T &&trap);
    //for lvalue
    void push(T &trap);
    size_t getSize();
    bool empty();
    T pop();

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

    template <class A>
    friend std::ostream& operator<<(std::ostream &os, const TStack<A>& stack);

    ~TStack();

```

```
private:
    std::shared_ptr<TStackItem<T>> top;
    size_t size;
};
```

```
#endif //LAB_2_TSTACK_H
```

- TStack.cpp

```
template <class T> TIterator<TStackItem<T>,T> TStack<T>::begin()
{
    return TIterator<TStackItem<T>,T>( top );
}

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

Выводы: в данной лабораторной работе я получил навыки программирования итераторов на языке C++, закрепил навык работы с шаблонами классов. Добавил в контейнеру возможность навигации при помощи итератора.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_5

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №6

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

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.

Теория

Алокатор умеет выделять и освобождать память в требуемых количествах определённым образом. `std::allocator` -- пример реализации аллокатора из стандартной библиотеки, просто использует `new` и `delete`, которые обычно обращаются к системным вызовам `malloc` и `free`.

Программист обладает преимуществом над стандартным аллокатором, он знает какое количество памяти будет выделяться чаще, как она будет связана.

Хорошим способом оптимизации программы будет уменьшение количества системных вызовов, которые происходят при аллокации. Аллоцировав сразу большой отрезок памяти и распределяя его, можно добиться положительных эффектов.

Листинг

- `TAllocationBlock.h`

```
#ifndef TALLOCATIONBLOCK_H
#define TALLOCATIONBLOCK_H

#include <cstdlib>
#include "TStackSnd.h"

#include "TStackSnd.h"

class TAllocationBlock {
public:
    TAllocationBlock(size_t size, size_t count);
    void *allocate();
    void deallocate(void *pointer);
    bool has_free_blocks();
};
```

```

    virtual ~TAllocationBlock();
private:
    size_t _size;
    size_t _count;

    char *_used_blocks;

    TStackSnd<void*> free_blocks_stack;

    size_t _free_count;

};

#endif      /* TALLOCATIONBLOCK_H */

```

- TAllocationBlock.cpp

```

#include "TAllocationBlock.h"

#include <iostream>

TAllocationBlock::TAllocationBlock(size_t size, size_t count):
    _size(size), _count(count) {
    _used_blocks = (char*)malloc(_size*_count);
    //_free_blocks = (void**)malloc(sizeof(void*)*_count);

    for(size_t i=0; i<_count; i++) {
        //_free_blocks[i] = _used_blocks+i*_size;
        free_blocks_stack.push(_used_blocks+i*_size);
    }
    _free_count = _count;
    std::cout << "TAllocationBlock: Memory init" << std::endl;
}

void *TAllocationBlock::allocate() {
    void *result = nullptr;

    if(_free_count>0)
    {
        result = free_blocks_stack.pop();
        //_result = _free_blocks[_free_count-1];
        _free_count--;
        std::cout << "TAllocationBlock: Allocate " << (_count-_free_count) << "
of " << _count << std::endl;
    } else
    {
        std::cout << "TAllocationBlock: No memory exception :-)" << std::endl;
        throw 1;
    }

    return result;
}

void TAllocationBlock::deallocate(void *pointer) {
    std::cout << "TAllocationBlock: Deallocate block " << std::endl;

    free_blocks_stack.push(pointer);
    //_free_blocks[_free_count] = pointer;
    _free_count ++;
}

```

```

}

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

TAllocationBlock::~TAllocationBlock() {

    if(_free_count<_count) std::cout << "TAllocationBlock: Memory leak?" <<
std::endl;
        else std::cout << "TAllocationBlock: Memory freed" <<
std::endl;
    //delete _free_blocks;
    delete _used_blocks;
}

```

- TStackItem.h

```

#ifndef LAB_2_TSTACKITEM_H
#define LAB_2_TSTACKITEM_H

#include "../figure/Trapeze.h"
#include "../figure/Pentagon.h"
#include "../figure/Rhombus.h"
#include "TAllocationBlock.h"

template <class T>
class TStackItem {
public:
    //type conversion
    TStackItem(const T &fig);
    //copy construct
    TStackItem(const TStackItem& orig);

    template <class A>
    friend std::ostream& operator<<(std::ostream& os, const TStackItem<A>&
item);

    void SetPrev(std::shared_ptr<TStackItem<T>> prev);
    std::shared_ptr<TStackItem<T>> GetPrev();
    T GetFigure() const;

    void * operator new (size_t size);
    void operator delete(void *p);

    virtual ~TStackItem();
private:
    static TAllocationBlock allocator;
    T figure;
    std::shared_ptr<TStackItem<T>> prev;
};

#endif //LAB_2_TSTACKITEM_H

```

- TStackItem.cpp

```

#include "TStackItem.h"

template <class T>
TStackItem<T>::TStackItem(const T &fig) {
    figure = fig;
    prev = nullptr;
    //std::cout << "Stack item: created" << std::endl;
}

```

```

}

template <class T>
TStackItem<T>::TStackItem(const TStackItem& orig) {
    figure = orig.figure;
    prev = orig.prev;
    //std::cout << "Stack item: copied" << std::endl;
}

template<class T>
TAllocationBlock TStackItem<T>::allocator(
    sizeof(TStackItem<T> ), 20);

template<class T>
void * TStackItem<T>::operator new(size_t size) {
    std::cout<<"//TStackItem: new item"<<std::endl;
    return allocator.allocate();
}

template<class T>
void TStackItem<T>::operator delete(void *p) {
    std::cout<<"//TStackItem: item deleted"<<std::endl;
    allocator.deallocate(p);
}

template <class T>
void TStackItem<T>::SetPrev(std::shared_ptr<TStackItem<T>> prev) {
    if(this != nullptr)
        this->prev = prev;
}

template <class T>
std::shared_ptr<TStackItem<T>> TStackItem<T>::GetPrev() {
    return this->prev;
}

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

template <class T>
TStackItem<T>::~TStackItem() {
    //std::cout << "Stack item: deleted" << std::endl;
}

template <class T>
std::ostream& operator<<(std::ostream& os, const TStackItem<T>& item) {
    std::cout << *item.figure;
    return os;
}

template class TStackItem<std::shared_ptr<Figure>>;
template std::ostream& operator<<(std::ostream &os, const
TStackItem<std::shared_ptr<Figure>>& stack);

    • TStackItemSnd.h
//
// TStackItemSnd.h
// 07.12.18

```

```
//
#ifndef LAB_4_TSTACKITEMSND_H
#define LAB_4_TSTACKITEMSND_H

#include <memory>

template <class T>
class TStackItemSnd {
public:
    //type conversion
    TStackItemSnd(const T &fig);
    //copy construct
    TStackItemSnd(const TStackItemSnd& orig);

    void SetPrev(std::shared_ptr<TStackItemSnd<T>> prev);
    std::shared_ptr<TStackItemSnd<T>> GetPrev();
    T GetFigure() const;

    virtual ~TStackItemSnd();
private:
    T figure;
    std::shared_ptr<TStackItemSnd<T>> prev;
};

#endif //LAB_4_TSTACKITEMSND_H
```

- TStackItemSnd.cpp

```
#include "TStackItemSnd.h"

template <class T>
TStackItemSnd<T>::TStackItemSnd(const T &fig) {
    figure = fig;
    prev = nullptr;
    //std::cout << "Stack item: created" << std::endl;
}

template <class T>
TStackItemSnd<T>::TStackItemSnd(const TStackItemSnd& orig) {
    figure = orig.figure;
    prev = orig.prev;
    //std::cout << "Stack item: copied" << std::endl;
}

template <class T>
void TStackItemSnd<T>::SetPrev(std::shared_ptr<TStackItemSnd<T>> prev) {
    if(this != nullptr)
        this->prev = prev;
}

template <class T>
std::shared_ptr<TStackItemSnd<T>> TStackItemSnd<T>::GetPrev() {
    return this->prev;
}

template <class T>
T TStackItemSnd<T>::GetFigure() const{
    return this->figure;
}
```



```
}  
  
template <class T>  
TStackItemSnd<T>::~~TStackItemSnd() {  
    //std::cout << "Stack item: deleted" << std::endl;  
}  
  
template class TStackItemSnd<void*>;
```

Выводы: в данной лабораторной работе я закрепил навыки работы с памятью на языке C++, получил навыки создания аллокаторов. Добавил аллокатор и упрощенный стек, в котором будут храниться адреса использованных/свободных блоков.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_6

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №7

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

Цель работы

Целью лабораторной работы является:

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

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.

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 индексу 0.
4. Добавляем Объект3: Объект добавляется в список, находящийся в массиве по индексу 0.
5. Добавляем Объект4: Объект добавляется в список, находящийся в массиве по индексу 0.
6. Добавляем Объект5: Объект добавляется в список, находящийся в массиве по индексу 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

Листинг

```
• TRemoveCriteria.h
#ifndef STACKS_TREMOVECRITERIA_H_
#define STACKS_TREMOVECRITERIA_H_
#include <memory>
```

```

template<class T>
class TRemoveCriteria {
public:
    virtual bool isIt(T& obj) = 0;
    virtual ~TRemoveCriteria() {};
private:
};

#endif /* STACKS_TREMOVECRITERIA_H_ */

```

- TRemoveCriteriaAll.h

```

#ifndef TSTACKS_TREMOVECRITERIAALL_H_
#define TSTACKS_TREMOVECRITERIAALL_H_

#include "TRemoveCriteria.h"
template<class T>
class TRemoveCriteriaAll: public TRemoveCriteria<T> {
public:
    TRemoveCriteriaAll() { }
    ;
    bool isIt(T& obj) override {
        return true;
    }
private:
};

#endif /* TSTACKS_TREMOVECRITERIAALL_H_ */

```

- TRemoveCriteriaByType.h

```

#ifndef TSTACKS_TREMOVECRITERIABYTYPE_H_
#define TSTACKS_TREMOVECRITERIABYTYPE_H_

#include "TRemoveCriteria.h"
template<class T>
class TRemoveCriteriaByType: public TRemoveCriteria<T> {
public:
    TRemoveCriteriaByType(double value) :
        _value(value) {
    }
    ;
    bool isIt(T& obj) override {
        return _value == obj->getType();
    }
private:
    char _value;
};

#endif /* TSTACKS_TREMOVECRITERIABYTYPE_H_ */

```

- TRemoveCriteriaByValue.h

```

#ifndef TSTACKS_TREMOVECRITERIABYVALUE_H_
#define TSTACKS_TREMOVECRITERIABYVALUE_H_

#include "TRemoveCriteria.h"

template<class T>
class TRemoveCriteriaByValue: public TRemoveCriteria<T> {
public:
    TRemoveCriteriaByValue(double value) :
        _value(value) {
    }
}

```

```

        ;
        bool isIt(T& obj) override {
            return _value == obj->Square();
        }
private:
    double _value;
};
#endif /* TSTACKS_TREMOVECRITERIABYVALUE_H_ */

```

• TStackTop.h

```

#ifndef LAB_4_TSTACKTOP_H
#define LAB_4_TSTACKTOP_H

#include "TStackItem.h"
#include "TIterator.h"
#include "TRemoveCriteria.h"

template <class T, class TT>
class TStackTop {

public:
    TStackTop();

    void InsertSubitem(TT &value);
    void InsertSubitem(TT &&value);

    void RemoveSubitem(TRemoveCriteria<TT> * criteria);

    //for rvalue
    void push(T &&elem);
    //for lvalue
    void push(T &elem);
    size_t getSize();
    bool empty();
    T pop();

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

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

    ~TStackTop();

private:
    std::shared_ptr<TStackItem<T>> top;
    size_t size;
};

#endif //LAB_4_TSTACKTOP_H

```

• TStackTop.cpp

```

#include "TStackTop.h"
#include "TStackBottom.h"

template <class T, class TT>
TStackTop<T, TT>::TStackTop() : top(nullptr), size(0) {
}

template <class T, class TT>

```

```

size_t TStackTop<T, TT>::getSize() {
    return this->size;
}

template <class T, class TT>
void TStackTop<T, TT>::push(T &&elem) {
    std::shared_ptr<TStackItem<T>> newItem(new TStackItem<T>(elem));
    newItem->SetPrev(top);
    top = newItem;
    size++;
}

template <class T, class TT>
void TStackTop<T, TT>::push(T &elem) {
    std::shared_ptr<TStackItem<T>> newItem(new TStackItem<T>(elem));
    newItem->SetPrev(top);
    top = newItem;
    size++;
}

template <class T, class TT>
bool TStackTop<T, TT>::empty() {
    return top == nullptr;
}

template <class T, class TT>
T TStackTop<T, TT>::pop() {
    T result;
    if (top != nullptr) {
        std::shared_ptr<TStackItem<T>> old_top = top;
        top = top->GetPrev();
        result = old_top->GetValue();
        old_top->SetPrev(nullptr);
        size--;
    }
    else {
        std::cout << "Stack isn't have elements" << std::endl;
    }

    return result;
}

template <class T, class TT>
void TStackTop<T, TT>::InsertSubitem(TT &value) {

    bool inserted = false;
    if (top != nullptr) {

        for (auto i : * this) {
            if (i->getSize() < 5) {
                T t_value;
                t_value =
std::make_shared<TStackBottom<std::shared_ptr<Figure>>>();
                double sq = value->Square();
                while(!i->empty()) {
                    TT elem = i->pop();
                    if(elem->Square() < sq) {
                        i->push(elem);
                        break;
                    }
                    t_value->push(elem);
                }
                i->push(value);
                while (!t_value->empty()) {

```

```

        TT elem = t_value->pop();
        i->push(elem);
    }

    //std::cout << "Stack: Add Element in stack:" << i->getSize() <<
std::endl;
    inserted = true;
}
}

if (!inserted) {
    //std::cout << "Stack: New stack element created" << std::endl;
    T t_value;
    t_value = std::make_shared<TStackBottom<std::shared_ptr<Figure>>>();
    t_value->push(value);
    push(t_value);
}

}

template <class T, class TT>
void TStackTop<T, TT>::InsertSubitem(TT &&value) {

    bool inserted = false;
    if (top != nullptr) {

        for (auto i : * this) {
            if (i->getSize() < 5) {
                T t_value;
                t_value =
std::make_shared<TStackBottom<std::shared_ptr<Figure>>>();
                double sq = value->Square();
                while(!i->empty()) {
                    TT elem = i->pop();
                    if(elem->Square() < sq) {
                        i->push(elem);
                        break;
                    }
                    t_value->push(elem);
                }
                i->push(value);
                while (!t_value->empty()) {
                    TT elem = t_value->pop();
                    i->push(elem);
                }

                //std::cout << "Stack: Add Element in stack:" << i->getSize() <<
std::endl;
                inserted = true;
            }
        }

        if (!inserted) {
            //std::cout << "Stack: New stack element created" << std::endl;
            T t_value;
            t_value = std::make_shared<TStackBottom<std::shared_ptr<Figure>>>();
            t_value->push(value);
            push(t_value);
        }
    }

}

template <class T, class TT>
void TStackTop<T, TT>::RemoveSubitem(TRemoveCriteria<TT> * criteria) {

```

```

std::cout << "----->" << std::endl;
for (auto i : * this) {
    T copy;
    copy = std::make_shared<TStackBottom<std::shared_ptr<Figure>>>();
    while (!i->empty()) {
        TT value = i->pop();

        if (criteria->isIt(value))
            std::cout << "Stack: Delete element " << std::endl << *value <<
std::endl;
        else {
            copy->push(value);
        }
    }

    while (!copy->empty()) {
        TT elem = copy->pop();
        i->push(elem);
    }
}
TStackTop<T, TT> copy;

while( !this->empty() ) {
    T elem = this->pop();
    if( !elem->empty() ) {
        copy.push(elem);
    }
}

while( !copy.empty() ) {
    T elem = copy.pop();
    this->push(elem);
}

std::cout << "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" << std::endl;
}

template <class T, class TT>
TStackTop<T, TT>::~~TStackTop() {
    while(!this->empty()) {
        this->pop();
    }
}

template <class T, class TT>
std::ostream& operator<<(std::ostream &os, const TStackTop<T, TT>& stack) {
    std::shared_ptr<TStackItem<T>>>item = stack.top;
    os << "Stack 1st lvl: " << std::endl;

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

template <class T, class TT>
TIterator<TStackItem<T>,T> TStackTop<T, TT>::begin()
{
    return TIterator<TStackItem<T>,T>( top );
}

```



```

}

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

#include "TStackBottom.h"

template class TStackTop<std::shared_ptr<TStackBottom<std::shared_ptr<Figure>>>,
std::shared_ptr<Figure>>>;
template std::ostream& operator<<(std::ostream &os, const
TStackTop<std::shared_ptr<TStackBottom<std::shared_ptr<Figure>>>,
std::shared_ptr<Figure>>>& stack);

```

- main.cpp

```

#include <cstdlib>
#include <iostream>
#include "figure/Figure.h"
#include "stack/TStackBottom.h"
#include "stack/TStackTop.h"
#include "stack/TRemoveCriteriaAll.h"
#include "stack/TRemoveCriteriaByValue.h"
#include "stack/TRemoveCriteriaByType.h"

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

    TStackTop<std::shared_ptr<TStackBottom<std::shared_ptr<Figure>>>,
std::shared_ptr<Figure>>> stackTop;

    unsigned int action;

    while (true) {
        std::cout << "Menu:" << std::endl;
        std::cout << "1) Add figure" << std::endl;
        std::cout << "2) Delete figures with square = <x>" << std::endl;
        std::cout << "3) Delete figures with type = <y>" << std::endl;
        std::cout << "4) Delete all figures" << std::endl;
        std::cout << "5) Print" << std::endl;
        std::cout << "6) Print size" << std::endl;
        std::cout << "0) Quit" << std::endl;
        std::cin >> action;

        if (action == 0) {
            break;
        }

        switch (action) {
            case 1: {
                char figure_type;
                std::cout
                    << "Enter figure type: t - trapeze, p - pentagon, r -
rhombus: ";

                std::cin >> figure_type;
                std::shared_ptr<Figure> t;
                switch (figure_type) {
                    case 't': {

```

```

        t = std::make_shared<Trapeze>(std::cin);
        stackTop.InsertSubitem(t);
        break;
    }
    case 'p': {
        t = std::make_shared<Pentagon>(std::cin);
        stackTop.InsertSubitem(t);
        break;
    }
    case 'r': {
        t = std::make_shared<Rhombus>(std::cin);
        stackTop.InsertSubitem(t);
        break;
    }
    default: {
        std::cout << "Try more..." << std::endl;
        break;
    }
}

break;
}
case 2: {
    int square;
    std::cout << "Enter square:\n";
    std::cin >> square;
    TRemoveCriteriaByValue<std::shared_ptr<Figure>>
criteria(square);
    stackTop.RemoveSubitem(&criteria);
    break;
}
case 3: {
    char figure_type;
    std::cout
        << "Enter figure type: t - trapeze, p - pentagon, r -
rhombus: ";
    std::cin >> figure_type;
    TRemoveCriteriaByType<std::shared_ptr<Figure>>
criteria(figure_type);
    stackTop.RemoveSubitem(&criteria);
    break;
}
case 4: {
    TRemoveCriteriaAll<std::shared_ptr<Figure>> criteria;
    stackTop.RemoveSubitem(&criteria);
    break;
}

case 5: {
    std::cout << stackTop << std::endl;
    break;
}

case 6: {
    size_t size = stackTop.getSize();
    std::cout << "size: " << size << std::endl;
    break;
}
default:
    std::cout << "Invalid action" << std::endl;
    break;
}
}

```

```
    return 0;  
}
```

Выводы: в данной лабораторной работе я закрепил навыки работы с памятью на языке C++, получил навыки создания сложных динамических структур. Создал сложное хранилище данных: стек со стеками, сортируемый по площади, с возможностью удаления по критериям.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_7

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №8

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

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

Теория

Параллельные вычисления — способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно). Термин охватывает совокупность вопросов параллелизма в программировании, а также создание эффективно действующих аппаратных реализаций. Теория параллельных вычислений составляет раздел прикладной теории алгоритмов.

Существуют различные способы реализации параллельных вычислений. Например, каждый вычислительный процесс может быть реализован в виде процесса операционной системы, либо же вычислительные процессы могут представлять собой набор потоков выполнения внутри одного процесса ОС. Параллельные программы могут физически исполняться либо последовательно на единственном процессоре — перемежая по очереди шаги выполнения каждого вычислительного процесса, либо параллельно — выделяя каждому вычислительному процессу один или несколько процессоров (находящихся рядом или распределённых в компьютерную сеть).

Быстрая сортировка относится к алгоритмам «разделяй и властвуй».

Алгоритм состоит из трёх шагов:

1. Выбрать элемент из массива. Назовём его опорным.
2. *Разбиение*: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один или отсутствуют элементы.

Листинг

```
• TStack.h
#ifndef LAB_2_TSTACK_H
#define LAB_2_TSTACK_H

#include "TStackItem.h"
```

```

#include "TIterator.h"
#include <memory>
#include <future>
#include <mutex>

template <class T>
class TStack {
public:
    TStack();
    //copy constructor
    TStack(const TStack<T>& orig);

    //for rvalue
    void push(T &&trap);
    //for lvalue
    void push(T &trap);
    size_t getSize();
    bool empty();
    T pop();
    T pop_last();

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

    void sort();
    void sort_parallel();

    template <class A>
    friend std::ostream& operator<<(std::ostream &os, const TStack<A>& stack);

    ~TStack();

private:
    std::future<void> sort_in_background();
    std::shared_ptr<TStackItem<T>> top;
    size_t size;
};

#endif //LAB_2_TSTACK_H

```

- TStack.cpp

```

#include "TStack.h"
#include <functional>

template <class T>
TStack<T>::TStack() : top(nullptr), size(0) {
}

template <class T>
void TStack<T>::sort() {
    if (getSize() > 1) {
        T middle = pop();
        TStack<T> left, right;

        while (!empty()) {
            T item = pop();
            if (*item < *middle) {
                left.push(item);
            } else {
                right.push(item);
            }
        }
        left.sort_parallel();
        right.sort_parallel();
        left.push(middle);
    }
}

```

```

        right.push(item);
    }
}

left.sort();
right.sort();

while (!left.empty()) push(left.pop_last());
push(middle);
while (!right.empty()) push(right.pop_last());
}
}

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

template <class T>
void TStack<T>::sort_parallel() {
    if (getSize() > 1) {
        T middle = pop_last();
        TStack<T> left, right;

        while (!empty()) {
            T item = pop_last();
            if (*item < *middle) {
                left.push(item);
            } else {
                right.push(item);
            }
        }

        std::future<void> left_res = left.sort_in_background();
        std::future<void> right_res = right.sort_in_background();

        left_res.get();

        while (!left.empty()) push(left.pop_last());
        push(middle);

        right_res.get();
        while (!right.empty()) push(right.pop_last());
    }
}
}

```

```

template <class T>
TStack<T>::TStack(const TStack<T>& orig) {
    std::shared_ptr<TStackItem<T>> item = orig.top;
    size = orig.size;
}

```

```

        while(item != nullptr) {
            item = item->GetPrev();
            this->push(item->GetFigure());
        }
    }

    template <class T>
    size_t TStack<T>::getSize() {
        return this->size;
    }

    template <class T>
    void TStack<T>::push(T &&trap) {
        std::shared_ptr<TStackItem<T>> newItem(new TStackItem<T>(trap));
        newItem->SetPrev(top);
        top = newItem;
        size++;
    }

    template <class T>
    void TStack<T>::push(T &trap) {
        std::shared_ptr<TStackItem<T>> newItem(new TStackItem<T>(trap));
        newItem->SetPrev(top);
        top = newItem;
        size++;
    }

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

    template <class T>
    T TStack<T>::pop() {
        T result;
        if (top != nullptr) {
            std::shared_ptr<TStackItem<T>>old_top = top;
            top = top->GetPrev();
            result = old_top->GetFigure();
            old_top->SetPrev(nullptr);
            size--;
        }
        else {
            std::cout << "Stack isn't have elements" << std::endl;
        }

        return result;
    }

    template <class T>
    T TStack<T>::pop_last() {
        T result;

        if (top != nullptr) {
            std::shared_ptr<TStackItem < T>> element = top;
            std::shared_ptr<TStackItem < T>> prev = nullptr;

            while (element->GetPrev() != nullptr) {
                prev = element;
                element = element->GetPrev();
            }

            if (prev != nullptr) {

```



```

        prev->SetPrev(nullptr);
        result = element->GetFigure();

    } else {
        result = element->GetFigure();
        top = nullptr;
    }
}

return result;
}

template <class T>
TStack<T>::~~TStack() {
    while(!this->empty()) {
        this->pop();
    }
}

template <class T>
std::ostream& operator<<(std::ostream &os, const TStack<T>& stack) {
    std::shared_ptr<TStackItem<T>> item = stack.top;
    os << "Stack: " << std::endl << std::endl;

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

template <class T> TIterator<TStackItem<T>,T> TStack<T>::begin()
{
    return TIterator<TStackItem<T>,T>( top );
}

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

template class TStack<std::shared_ptr<Figure>>;
template std::ostream& operator<<(std::ostream &os, const
TStack<std::shared_ptr<Figure>>& stack);

```

Выводы: в данной лабораторной работе я получил алгоритмы работы с параллельным программированием в C++. Реализовал алгоритм быстрой сортировки и распараллелил его.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_8

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра 806 «Вычислительная математика и программирование»

Объектно-ориентированное программирование

Лабораторная работа №9

Студент:	Дубинин А. О.
Год приёма:	2018
Группа:	М8О-206Б
Преподаватель:	Поповкин А. В.
Вариант:	№36

Москва, 2018

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 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.

Для создания потоков использовать механизмы:

- future
- packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

- mutex
- lock_guard

Теория:

Лямбда-выражение в программировании — специальный синтаксис для определения функциональных объектов, заимствованный из λ -исчисления. Применяется как правило для объявления анонимных функций по месту их использования, и обычно допускает замыкание на лексический контекст, в котором это выражение использовано. Используя лямбда-выражения, можно объявлять функции в любом месте кода.

Листинг

- main.c

```
#include <cstdlib>
#include <iostream>
#include <future>
#include <functional>
#include <random>
#include <thread>
#include <ctime>
#include "figure/Figure.h"
#include "stack/TStack.h"
```

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

```
    srand(time(NULL));
```

```
    TStack<std::shared_ptr<Figure>> stack;
    typedef std::function<void (void) > command;
```

```

TStack <std::shared_ptr<command>> stack_cmd;

command cmd_insert = [&]() {
    std::cout << "Command: Insert figure with random sides" << std::endl;

    int fst_value = (rand() %1000) + 1;
    int snd_value = (rand() %1000) + 1;
    int trd_value = (rand() %1000) + 1;
    std::shared_ptr<Figure> t;
    int action = (rand() %3) + 1;
    switch (action) {
        case 1:
            t = std::make_shared<Pentagon>(fst_value, snd_value);
            break;
        case 2:
            t = std::make_shared<Rhombus>(fst_value, snd_value);
            break;
        case 3:
            t = std::make_shared<Trapeze>(fst_value, snd_value, trd_value);
            break;
    }

    stack.push(t);
};

command cmd_print = [&]() {
    std::cout << "Command: Print stack" << std::endl;
    std::cout << stack;
};

command cmd_delete = [&]() {
    int number;
    std::cout << "Enter the number(delete lambda)"<<std::endl;
    std::cin >> number;
    TStack<std::shared_ptr<Figure>> tmp;
    while(!stack.empty()) {
        auto figure = stack.pop();
        if(figure->Square() >= number) {
            tmp.push(figure);
        }
    }
    while (!tmp.empty()) {
        auto figure = tmp.pop();
        stack.push(figure);
    }
};

unsigned int action;

while (true) {
    std::cout << "Menu:" << std::endl;
    std::cout << "1) Add command generate" << std::endl;
    std::cout << "2) Add command print" << std::endl;
    std::cout << "3) Add command delete" << std::endl;
    std::cout << "4) Add command execute" << std::endl;
    std::cout << "5) Print" << std::endl;
    std::cout << "0) Quit" << std::endl;
    std::cin >> action;

    if (action == 0) {
        break;
    }
}

```

```

        switch (action) {
            case 1: {
                stack_cmd.push(std::shared_ptr<command> (&cmd_insert, []
(command*) {
                    }));
                break;
            }
            case 2: {
                stack_cmd.push(std::shared_ptr<command> (&cmd_print, []
(command*) {
                    }));
                break;
            }
            case 3: {
                stack_cmd.push(std::shared_ptr<command> (&cmd_delete, []
(command*) {
                    }));
                break;
            }
            case 4: {
                while (!stack_cmd.empty()) {
                    std::shared_ptr<command> cmd = stack_cmd.pop();
                    std::future<void> ft = std::async(*cmd);
                    ft.get();
                }
                break;
            }
            case 5: {
                for(auto i : stack)
                    std::cout << *i << std::endl;
                break;
            }
            default:
                std::cout << "Invalid action" << std::endl;
                break;
        }
    }

    return 0;
}

```

Выводы: в данной лабораторной работе я получил навыки работы с лямбда-выражениями. Добавил с помощью лямбда-выражений команды, выполняющие операции с контейнером 1-го уровня: заполнение его случайно сгенерированными фигурами, вывод контейнера на экран, удаление элементов с площадью меньше заданной.

Ссылка на код: https://github.com/ArtDu/oop/tree/master/lab_9