Московский Авиационный Институт (Национальный Исследовательский Университет)

**Факультет прикладной математики и физики**

Кафедра вычислительной математики и программирования

**Отчет по лабораторным работам 1-9, по предмету «Объектно-ориентированное программирование».**

Студент: Графчикова Я.А. Группа: 08-204, № по списку 3

Руководитель: Поповкин А.В.

Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Москва, 2017**

**Вариант №3 :**

**Контейнер 1-ого уровня : массив**

**Контейнер 2-ого уровня : бинарное дерево**

**Фигура №1 : прямоугольник**

**Фигура №2 : ромб**

**Фигура №3 : трапеция**

**ЛАБОРАТОРНАЯ РАБОТА №1**

ЦЕЛЬ РАБОТЫ

• Программирование классов на языке С++

• Управление памятью в языке С++

• Изучение базовых понятий ООП.

• Знакомство с классами в C++.

• Знакомство с перегрузкой операторов.

• Знакомство с дружественными функциями.

• Знакомство с операциями ввода-вывода из стандартных библиотек.

ЗАДАНИЕ

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

Классы должны удовлетворять следующим правилам:

• Должны иметь общий родительский класс Figure.

• Должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в

стандартный поток вывода cout.

• Должный иметь общий виртуальный метод расчета площади фигуры – Square.

• Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного

потока cin.

• Должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание

методов (.cpp).

Программа должна позволять вводить фигуру каждого типа с клавиатуры, выводить параметры фигур на

экран и их площадь.

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

bool err=false;

virtual void Print() = 0;

virtual double Square() = 0;

virtual ~Figure() {}

};

#endif

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure{

public:

Rectangle();

Rectangle(std::istream& is);

void Print() override;

double Square() override;

private:

double side\_a;

double side\_b;

};

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(std::istream& is);

void Print() override;

double Square() override;

private:

double side;

double height;

};

#endif

**Trapezium.h**

#ifndef TRAPEZIUM\_H

#define TRAPEZIUM\_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure{

public:

Trapezium();

Trapezium(std::istream& is);

void Print() override;

double Square() override;

private:

double side\_a;

double side\_b;

double height;

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>side\_a)){

is.clear();

while(is.get() !='\n');

err=true;

return;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>side\_b)){

is.clear();

while(is.get() !='\n');

err=true;

return;

}

if(side\_a<0 || side\_b<0) err=true;

}

void Rectangle::Print(){

std::cout<<"Тип фигуры: прямоугольник"<<std::endl;

std::cout<<"Сторона a= "<<side\_a<<std::endl;

std::cout<<"Сторона b= "<<side\_b<<std::endl;

}

double Rectangle::Square(){

return side\_a\*side\_b;

}

**Rhombus.cpp**

#include "Rhombus.h"

Rhombus::Rhombus(){

side=0.0;

height=0.0;

}

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

std::cout<<"Введите сторону: "<<std::endl;

if(!(is>>side)){

is.clear();

while(is.get() !='\n');

err=true;

return;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>height)){

is.clear();

while(is.get() !='\n');

err=true;

return;

}

if(height>side){

err=true;

return;

}

}

void Rhombus::Print(){

std::cout<<"Тип фигуры: ромб"<<std::endl;

std::cout<<"Сторона= "<<side<<std::endl;

std::cout<<"Высота= "<<height<<std::endl;

}

double Rhombus::Square(){

return side\*height;

}

**Trapezium.cpp**

#include "Trapezium.h"

Trapezium::Trapezium(){

side\_a=0.0;

side\_b=0.0;

height=0.0;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>side\_a)){

is.clear();

while(is.get() !='\n');

err=true;

return;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>side\_b)){

is.clear();

while(is.get() !='\n');

err=true;

return;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>height)){

is.clear();

while(is.get() !='\n');

err=true;

return;

}

if(side\_a<0 || side\_b<0 || height<0) err=true;

}

void Trapezium::Print(){

std::cout<<"Тип фигуры: трапеция"<<std::endl;

std::cout<<"Сторона a= "<<side\_a<<std::endl;

std::cout<<"Сторона b= "<<side\_b<<std::endl;

std::cout<<"Высота= "<<height<<std::endl;

}

double Trapezium::Square(){

return (side\_a+side\_b)/2.0\*height;

}

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

void test(Figure\* figure){

if(figure->err){

std::cout<<"Некорректные входные данные"<<std::endl;

return;

}

figure->Print();

std::cout<<"Площадь фигуры= "<<figure->Square()<<std::endl;

delete figure;

}

int main()

{

setlocale(LC\_CTYPE,"rus");

int a;

do{

std::cout<<"1) Прямоугольник\n"<<"2) Трапеция\n"<<"3) Ромб\n"<<"4) Выход\n"<<"Выберете действие: "<<std::endl;

if(!(std::cin>>a)){

std::cin.clear();

while(std::cin.get() !='\n');

}

switch(a){

case 1:

test(new Rectangle(std::cin));

break;

case 2:

test(new Trapezium(std::cin));

break;

case 3:

test(new Rhombus(std::cin));

break;

case 4: break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

}while(a!=4);

return 0;

}

**Вывод:**

1) Прямоугольник

2) Трапеция

3) Ромб

4) Выход

Выберете действие:

1

Введите сторону a:

2

Введите сторону b:

1

Тип фигуры: прямоугольник

Сторона a= 2

Сторона b= 1

Площадь фигуры= 2

1) Прямоугольник

2) Трапеция

3) Ромб

4) Выход

Выберете действие:

2

Введите сторону a:

4

Введите сторону b:

3

Введите высоту:

2

Тип фигуры: трапеция

Сторона a= 4

Сторона b= 3

Высота= 2

Площадь фигуры= 7

1) Прямоугольник

2) Трапеция

3) Ромб

4) Выход

Выберете действие:

3

Введите сторону:

7

Введите высоту:

3

Тип фигуры: ромб

Сторона= 7

Высота= 3

Площадь фигуры= 21

1) Прямоугольник

2) Трапеция

3) Ромб

4) Выход

Выберете действие:

4

Process returned 0 (0x0) execution time : 40.757 s

Press any key to continue.

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 соджержать объекты фигур “по значению” (не по ссылке).

• Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.

• Класс-контейнер должен иметь методы по получению фигуры из контейнера (опеределяется

структурой контейнера).

• Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опеределяется

структурой контейнера).

• Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток

std::ostream (<<).

• Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.

• Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно

описание методов (.cpp).

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.

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

class Rectangle{

public:

bool err=false;

Rectangle();

Rectangle(std::istream & is);

double Square();

Rectangle& operator =(const Rectangle & right);

bool operator ==(const Rectangle & right) const;

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

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

virtual ~Rectangle();

private:

double side\_a;

double side\_b;

};

#endif

**Cont.h**

#ifndef CONT.H

#define CONT.H

#include "Rectangle.h"

#include "ContItem.h"

class Cont{

public:

Cont();

void push(Rectangle &rectangle);

Rectangle pop();

void removeIt();

ContItem\* getHead();

friend std::ostream& operator <<(std::ostream & os,const Cont & con);

virtual ~Cont();

private:

ContItem \*head;

};

#endif

**ContItem.h**

#ifndef CONTITEM\_H

#define CONTITEM\_H

#include "Rectangle.h"

class ContItem{

public:

ContItem(const Rectangle & rectangle);

//ContItem(const ContItem& orig);

ContItem\* SetNext(ContItem\* next);

ContItem\* GetNext();

Rectangle GetRectangle() const;

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

virtual ~ContItem();

private:

Rectangle rectangle;

ContItem \*next;

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

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

is>>\*this;

}

double Rectangle::Square(){

return side\_a\*side\_b;

}

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

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

side\_a = right.side\_a;

side\_b = right.side\_b;

//std::cout << "Прямоугольник скопирован" << std::endl;

return \*this;

}

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

return (side\_a == right.side\_a && side\_b == right.side\_b);

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0) obj.err=true;

return is;

}

Rectangle::~Rectangle(){

//std::cout << "Прямоугольник удален" << std::endl;

}

**Cont.cpp**

#include "Cont.h"

#include <iostream>

Cont::Cont(){

head=nullptr;

}

void Cont::push(Rectangle &rectangle){

ContItem \*other = new ContItem(rectangle);

other->SetNext(head);

head = other;

std::cout<<"Фигура добавлена"<<std::endl;

}

Rectangle Cont::pop(){

Rectangle result;

if (head != nullptr) {

ContItem \*old\_head = head;

head = head->GetNext();

result = old\_head->GetRectangle();

old\_head->SetNext(nullptr);

delete old\_head;

}

return result;

}

void Cont::removeIt(){

ContItem \*old\_head=head;

if(old\_head==nullptr){

std::cout<<"Стек пуст"<<std::endl;

return;

}

head=head->GetNext();

old\_head->SetNext(nullptr);

delete old\_head;

std::cout<<"Фигура удалена"<<std::endl;

}

ContItem\* Cont::getHead(){

return this->head;

}

std::ostream& operator <<(std::ostream & os,const Cont & con){

ContItem \*item = con.head;

if(item==nullptr) std::cout<<"Стек пуст"<<std::endl;

while(item!=nullptr)

{

os << \*item;

item = item->GetNext();

}

return os;

}

Cont::~Cont(){

delete head;

}

**ContItem.cpp**

#include "ContItem.h"

#include <iostream>

ContItem::ContItem(const Rectangle & rectangle){

this->rectangle = rectangle;

this->next = nullptr;

}

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

ContItem\* old = this->next;

this->next = next;

return old;

}

ContItem\* ContItem::GetNext(){

return this->next;

}

Rectangle ContItem::GetRectangle() const{

return this->rectangle;

}

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

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

return os;

}

ContItem::~ContItem(){

delete next;

}

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "ContItem.h"

#include "Cont.h"

int main()

{

setlocale(LC\_CTYPE,"rus");

Cont container;

int a;

do{

std::cout<<"1) Добавить фигуру\n2) Вывести содержимое\n3) Удалить фигуру\n4) Выход\n"<<std::endl;

if(!(std::cin>>a)){

std::cin.clear();

while(std::cin.get() !='\n');

}

switch(a){

case 1:{

Rectangle rect(std::cin);

if(rect.err){

std::cout<<"Некорректные входные данные"<<std::endl;

break;

}

container.push(rect);

break;}

case 2:

std::cout<<container;

break;

case 3:

container.removeIt();

break;

case 4:break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

}while(a!=4);

return 0;

}

**Вывод:**

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

1

Введите сторону a:

2

Введите сторону b:

3

Фигура добавлена

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

1

Введите сторону a:

5

Введите сторону b:

4

Фигура добавлена

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

2

[Сторона а= 5, Сторона b= 4]

[Сторона а= 2, Сторона b= 3]

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

3

Фигура удалена

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

2

[Сторона а= 2, Сторона b= 3]

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

4

Process returned 0 (0x0) execution time : 28.491 s

Press any key to continue.

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 (опеределяется

структурой контейнера).

• Класс-контейнер должен иметь метод по удалению фигуры из контейнера (опеределяется

структурой контейнера).

• Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток

std::ostream (<<).

• Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.

• Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно

описание методов (.cpp).

Нельзя использовать:

• Стандартные контейнеры std.

• Шаблоны (template).

• Объекты «по-значению»

Программа должна позволять:

• Вводить произвольное количество фигур и добавлять их в контейнер.

• Распечатывать содержимое контейнера.

• Удалять фигуры из контейнера.

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

bool err=false;

virtual ~Figure() {}

};

#endif

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure{

public:

Rectangle();

Rectangle(std::istream& is);

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

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

virtual ~Rectangle();

private:

double side\_a;

double side\_b;

};

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(std::istream& is);

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

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

virtual ~Rhombus();

private:

double side;

double height;

};

#endif

**Trapesium.h**

#ifndef TRAPEZIUM\_H

#define TRAPEZIUM\_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure{

public:

Trapezium();

Trapezium(std::istream& is);

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

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

virtual ~Trapezium();

private:

double side\_a;

double side\_b;

double height;

};

#endif

**Cont.h**

#ifndef CONT.H

#define CONT.H

#include <memory>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "ContItem.h"

const int size = 10;

class Cont{

public:

Cont();

void push(std::shared\_ptr<Rectangle> &&rectangle, int index);

void push(std::shared\_ptr<Rhombus> &&rhombus, int index);

void push(std::shared\_ptr<Trapezium> &&trapezium, int index);

bool IsRectangle(int index);

bool IsRhombus(int index);

bool IsTrapezium(int index);

std::shared\_ptr<Rectangle> GetRectangle(int index);

std::shared\_ptr<Rhombus> GetRhombus(int index);

std::shared\_ptr<Trapezium> GetTrapezium(int index);

void removeIt(int index);

friend std::ostream& operator <<(std::ostream & os,const Cont & cont);

virtual ~Cont();

private:

ContItem a[size];

};

#endif

**ContItem.h**

#ifndef CONTITEM\_H

#define CONTITEM\_H

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

class ContItem{

public:

ContItem();

ContItem(std::shared\_ptr<Rectangle> &rectangle);

ContItem(std::shared\_ptr<Rhombus> &rhombus);

ContItem(std::shared\_ptr<Trapezium> &trapezium);

bool IsRectangle() const;

bool IsRhombus() const;

bool IsTrapezium() const;

std::shared\_ptr<Rectangle> GetRectangle();

std::shared\_ptr<Rhombus> GetRhombus();

std::shared\_ptr<Trapezium> GetTrapezium();

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

virtual ~ContItem();

private:

std::shared\_ptr<Rectangle> rectangle;

std::shared\_ptr<Rhombus> rhombus;

std::shared\_ptr<Trapezium> trapezium;

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0) obj.err=true;

return is;

}

Rectangle::~Rectangle(){

//std::cout << "Прямоугольник удален" << std::endl;

}

**Rhombus.cpp**

#include "Rhombus.h"

#include <iostream>

Rhombus::Rhombus(){

side=0.0;

height=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона = "<<obj.side<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону: "<<std::endl;

if(!(is>>obj.side)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side<0 || obj.height<0) obj.err=true;

return is;

}

Rhombus::~Rhombus(){

//std::cout << "Ромб удален" << std::endl;

}

**Trapesium.cpp**

#include "Trapezium.h"

#include <iostream>

Trapezium::Trapezium(){

side\_a=0.0;

side\_b=0.0;

height=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0 || obj.height<0) obj.err=true;

return is;

}

Trapezium::~Trapezium(){

//std::cout << "Трапеция удален" << std::endl;

}

**Cont.cpp**

#include "Cont.h"

#include <iostream>

Cont::Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem();

}

void Cont::push(std::shared\_ptr<Rectangle> &&rectangle, int index){

a[index]=ContItem(rectangle);

}

void Cont::push(std::shared\_ptr<Rhombus> &&rhombus, int index){

a[index]=ContItem(rhombus);

}

void Cont::push(std::shared\_ptr<Trapezium> &&trapezium, int index){

a[index]=ContItem(trapezium);

}

bool Cont::IsRectangle(int index){

return a[index].IsRectangle();

}

bool Cont::IsRhombus(int index){

return a[index].IsRhombus();

}

bool Cont::IsTrapezium(int index){

return a[index].IsTrapezium();

}

std::shared\_ptr<Rectangle> Cont::GetRectangle(int index){

return a[index].GetRectangle();

}

std::shared\_ptr<Rhombus> Cont::GetRhombus(int index){

return a[index].GetRhombus();

}

std::shared\_ptr<Trapezium> Cont::GetTrapezium(int index){

return a[index].GetTrapezium();

}

void Cont::removeIt(int index){

a[index] = ContItem();

}

std::ostream& operator <<(std::ostream & os,const Cont & cont){

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

os << "[" << i << "] - " << cont.a[i] << std::endl;

}

return os;

}

Cont::~Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem();

}

**ContItem.cpp**

#include "ContItem.h"

ContItem::ContItem() : rectangle(nullptr), rhombus(nullptr), trapezium(nullptr) {}

ContItem::ContItem(std::shared\_ptr<Rectangle> &rectangle) : rectangle(rectangle), rhombus(nullptr), trapezium(nullptr) {}

ContItem::ContItem(std::shared\_ptr<Rhombus> &rhombus) : rectangle(nullptr), rhombus(rhombus), trapezium(nullptr) {}

ContItem::ContItem(std::shared\_ptr<Trapezium> &trapezium) : rectangle(nullptr), rhombus(nullptr), trapezium(trapezium) {}

ContItem::~ContItem() {}

bool ContItem::IsRectangle() const

{

if (rectangle != nullptr) return true;

else return false;

}

bool ContItem::IsRhombus() const

{

if (rhombus != nullptr) return true;

else return false;

}

bool ContItem::IsTrapezium() const

{

if (trapezium != nullptr) return true;

else return false;

}

std::shared\_ptr<Rectangle> ContItem::GetRectangle()

{

return this->rectangle;

}

std::shared\_ptr<Rhombus> ContItem::GetRhombus()

{

return this->rhombus;

}

std::shared\_ptr<Trapezium> ContItem::GetTrapezium()

{

return this->trapezium;

}

std::ostream& operator << (std::ostream &os, const ContItem &item)

{

if (item.IsRectangle())

os << \*item.rectangle << " (прямоугольник)";

else if (item.IsRhombus())

os << \*item.rhombus << " (ромб)";

else if (item.IsTrapezium())

os << \*item.trapezium << " (трапеция)";

else

os << "Пусто";

return os;

}

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

#include "ContItem.h"

#include "Cont.h"

int main()

{

setlocale(LC\_CTYPE,"rus");

Cont container;

int a;

do{

std::cout<<"1) Добавить фигуру\n2) Вывести содержимое\n3) Удалить фигуру\n4) Вывести фигуру\n5) Выход\n"<<std::endl;

if(!(std::cin>>a)){

std::cin.clear();

while(std::cin.get() !='\n');

}

switch(a){

case 1:{

std::cout << "1) Прямоугольник\n2) Ромб\n3) Трапеция\n" << std::endl;

int v,i;

std::cin >> v;

std::cout << "Введите индекс ";

std::cin >> i;

switch(v){

case 1:

container.push(std::shared\_ptr<Rectangle>(new Rectangle(std::cin)), i);

break;

case 2:

container.push(std::shared\_ptr<Rhombus>(new Rhombus(std::cin)), i);

break;

case 3:

container.push(std::shared\_ptr<Trapezium>(new Trapezium(std::cin)), i);

break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

break;}

case 2:

std::cout<<container;

break;

case 3:

std::cout << "Введите индекс ";

int ind;

std::cin >> ind;

container.removeIt(ind);

break;

case 4:

std::cout << "Введите индекс ";

int i;

std::cin >> i;

if (container.IsRectangle(i))

std::cout << \*container.GetRectangle(i) << std::endl;

else if (container.IsRhombus(i))

std::cout << \*container.GetRhombus(i) << std::endl;

else if (container.IsTrapezium(i))

std::cout << \*container.GetTrapezium(i) << std::endl;

else

std::cout << "Пустой элемент" << std::endl;

break;

case 5:break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

}while(a!=5);

return 0;

}

**Вывод:**

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 1

Введите сторону a:

3

Введите сторону b:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

2

Введите индекс 3

Введите сторону:

2

Введите высоту:

1

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

3

Введите индекс 0

Введите сторону a:

5

Введите сторону b:

3

Введите высоту:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

2

[0] - Сторона а= 5, Сторона b= 3, Высота = 2 (трапеция)

[1] - Сторона а= 3, Сторона b= 2 (прямоугольник)

[2] - Пусто

[3] - Сторона = 2, Высота = 1 (ромб)

[4] - Пусто

[5] - Пусто

[6] - Пусто

[7] - Пусто

[8] - Пусто

[9] - Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

3

Введите индекс 1

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

2

[0] - Сторона а= 5, Сторона b= 3, Высота = 2 (трапеция)

[1] - Пусто

[2] - Пусто

[3] - Сторона = 2, Высота = 1 (ромб)

[4] - Пусто

[5] - Пусто

[6] - Пусто

[7] - Пусто

[8] - Пусто

[9] - Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

4

Введите индекс 0

Сторона а= 5, Сторона b= 3, Высота = 2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

5

Process returned 0 (0x0) execution time : 40.879 s

Press any key to continue.

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.

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

bool err=false;

virtual ~Figure() {}

};

#endif

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure{

public:

Rectangle();

Rectangle(std::istream& is);

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

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

virtual ~Rectangle();

private:

double side\_a;

double side\_b;

};

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(std::istream& is);

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

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

virtual ~Rhombus();

private:

double side;

double height;

};

#endif

**Trapezium.h**

#ifndef TRAPEZIUM\_H

#define TRAPEZIUM\_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure{

public:

Trapezium();

Trapezium(std::istream& is);

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

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

virtual ~Trapezium();

private:

double side\_a;

double side\_b;

double height;

};

#endif

**Cont.h**

#ifndef CONT.H

#define CONT.H

#include <memory>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "ContItem.h"

const int size = 10;

template <class T1, class T2, class T3>

class Cont{

public:

Cont();

void push(std::shared\_ptr<T1> &&t, int index);

void push(std::shared\_ptr<T2> &&t, int index);

void push(std::shared\_ptr<T3> &&t, int index);

bool IsT1(int index);

bool IsT2(int index);

bool IsT3(int index);

std::shared\_ptr<T1> GetT1(int index);

std::shared\_ptr<T2> GetT2(int index);

std::shared\_ptr<T3> GetT3(int index);

void removeIt(int index);

template <class A1, class A2, class A3> friend std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont);

virtual ~Cont();

private:

ContItem<T1, T2, T3> a[size];

};

#endif

**ContItem.h**

#ifndef CONTITEM\_H

#define CONTITEM\_H

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

template <class T1, class T2, class T3>

class ContItem{

public:

ContItem();

ContItem(std::shared\_ptr<T1> &t);

ContItem(std::shared\_ptr<T2> &t);

ContItem(std::shared\_ptr<T3> &t);

bool IsT1() const;

bool IsT2() const;

bool IsT3() const;

std::shared\_ptr<T1> GetT1();

std::shared\_ptr<T2> GetT2();

std::shared\_ptr<T3> GetT3();

template <class A1, class A2, class A3> friend std::ostream& operator<< (std::ostream & os, const ContItem<A1, A2, A3> & item);

virtual ~ContItem();

private:

std::shared\_ptr<T1> t1;

std::shared\_ptr<T2> t2;

std::shared\_ptr<T3> t3;

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0) obj.err=true;

return is;

}

Rectangle::~Rectangle(){

//std::cout << "Прямоугольник удален" << std::endl;

}

**Rhombus.cpp**

#include "Rhombus.h"

#include <iostream>

Rhombus::Rhombus(){

side=0.0;

height=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона = "<<obj.side<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону: "<<std::endl;

if(!(is>>obj.side)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side<0 || obj.height<0) obj.err=true;

return is;

}

Rhombus::~Rhombus(){

//std::cout << "Ромб удален" << std::endl;

}

**Trapezium.cpp**

#include "Trapezium.h"

#include <iostream>

Trapezium::Trapezium(){

side\_a=0.0;

side\_b=0.0;

height=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0 || obj.height<0) obj.err=true;

return is;

}

Trapezium::~Trapezium(){

//std::cout << "Трапеция удален" << std::endl;

}

**Cont.cpp**

#include "Cont.h"

#include <iostream>

template <class T1, class T2, class T3> Cont<T1, T2, T3>::Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem<T1, T2, T3>();

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T1> &&t, int index){

a[index]=ContItem<T1, T2, T3>(t);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T2> &&t, int index){

a[index]=ContItem<T1, T2, T3>(t);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T3> &&t, int index){

a[index]=ContItem<T1, T2, T3>(t);

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT1(int index){

return a[index].IsT1();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT2(int index){

return a[index].IsT2();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT3(int index){

return a[index].IsT3();

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> Cont<T1, T2, T3>::GetT1(int index){

return a[index].GetT1();

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> Cont<T1, T2, T3>::GetT2(int index){

return a[index].GetT2();

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> Cont<T1, T2, T3>::GetT3(int index){

return a[index].GetT3();

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::removeIt(int index){

a[index] = ContItem<T1, T2, T3>();

}

template <class A1, class A2, class A3> std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont){

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

os << "[" << i << "] - " << cont.a[i] << std::endl;

}

return os;

}

template <class T1, class T2, class T3> Cont<T1, T2, T3>::~Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem<T1, T2, T3>();

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class Cont<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const Cont<Rectangle, Rhombus, Trapezium> & cont);

**ContItem.cpp**

#include "ContItem.h"

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem() : t1(nullptr), t2(nullptr), t3(nullptr) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T1> &t) : t1(t), t2(nullptr), t3(nullptr) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T2> &t) : t1(nullptr), t2(t), t3(nullptr) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T3> &t) : t1(nullptr), t2(nullptr), t3(t) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::~ContItem() {}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT1() const

{

if (t1 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT2() const

{

if (t2 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT3() const

{

if (t3 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> ContItem<T1, T2, T3>::GetT1()

{

return this->t1;

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> ContItem<T1, T2, T3>::GetT2()

{

return this->t2;

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> ContItem<T1, T2, T3>::GetT3()

{

return this->t3;

}

template <class A1, class A2, class A3> std::ostream& operator << (std::ostream &os, const ContItem<A1, A2, A3> &item)

{

if (item.IsT1())

os << \*item.t1 << " (прямоугольник)";

else if (item.IsT2())

os << \*item.t2 << " (ромб)";

else if (item.IsT3())

os << \*item.t3 << " (трапеция)";

else

os << "Пусто";

return os;

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class ContItem<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const ContItem<Rectangle, Rhombus, Trapezium> & item);

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

#include "ContItem.h"

#include "Cont.h"

int main()

{

setlocale(LC\_CTYPE,"rus");

Cont<Rectangle, Rhombus, Trapezium> container;

int a;

do{

std::cout<<"1) Добавить фигуру\n2) Вывести содержимое\n3) Удалить фигуру\n4) Вывести фигуру\n5) Выход\n"<<std::endl;

if(!(std::cin>>a)){

std::cin.clear();

while(std::cin.get() !='\n');

}

switch(a){

case 1:{

std::cout << "1) Прямоугольник\n2) Ромб\n3) Трапеция\n" << std::endl;

int v,i;

std::cin >> v;

std::cout << "Введите индекс ";

std::cin >> i;

switch(v){

case 1:

container.push(std::shared\_ptr<Rectangle>(new Rectangle(std::cin)), i);

break;

case 2:

container.push(std::shared\_ptr<Rhombus>(new Rhombus(std::cin)), i);

break;

case 3:

container.push(std::shared\_ptr<Trapezium>(new Trapezium(std::cin)), i);

break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

break;}

case 2:

std::cout<<container;

break;

case 3:

std::cout << "Введите индекс ";

int ind;

std::cin >> ind;

container.removeIt(ind);

break;

case 4:

std::cout << "Введите индекс ";

int i;

std::cin >> i;

if (container.IsT1(i))

std::cout << \*container.GetT1(i) << std::endl;

else if (container.IsT2(i))

std::cout << \*container.GetT2(i) << std::endl;

else if (container.IsT3(i))

std::cout << \*container.GetT3(i) << std::endl;

else

std::cout << "Пустой элемент" << std::endl;

break;

case 5:break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

}while(a!=5);

return 0;

}

**Вывод:**

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 1

Введите сторону a:

3

Введите сторону b:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

2

Введите индекс 3

Введите сторону:

2

Введите высоту:

1

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

3

Введите индекс 0

Введите сторону a:

5

Введите сторону b:

3

Введите высоту:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

2

[0] - Сторона а= 5, Сторона b= 3, Высота = 2 (трапеция)

[1] - Сторона а= 3, Сторона b= 2 (прямоугольник)

[2] - Пусто

[3] - Сторона = 2, Высота = 1 (ромб)

[4] - Пусто

[5] - Пусто

[6] - Пусто

[7] - Пусто

[8] - Пусто

[9] - Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

3

Введите индекс 1

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

2

[0] - Сторона а= 5, Сторона b= 3, Высота = 2 (трапеция)

[1] - Пусто

[2] - Пусто

[3] - Сторона = 2, Высота = 1 (ромб)

[4] - Пусто

[5] - Пусто

[6] - Пусто

[7] - Пусто

[8] - Пусто

[9] - Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

4

Введите индекс 0

Сторона а= 5, Сторона b= 3, Высота = 2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

5

Process returned 0 (0x0) execution time : 40.879 s

Press any key to continue.

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.

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

bool err=false;

virtual ~Figure() {}

};

#endif

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure{

public:

Rectangle();

Rectangle(std::istream& is);

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

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

virtual ~Rectangle();

private:

double side\_a;

double side\_b;

};

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(std::istream& is);

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

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

virtual ~Rhombus();

private:

double side;

double height;

};

#endif

**Trapezium.h**

#ifndef TRAPEZIUM\_H

#define TRAPEZIUM\_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure{

public:

Trapezium();

Trapezium(std::istream& is);

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

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

virtual ~Trapezium();

private:

double side\_a;

double side\_b;

double height;

};

#endif

**Cont.h**

#ifndef CONT.H

#define CONT.H

#include <memory>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "ContItem.h"

#include "Iterator.cpp"

const int size = 10;

template <class T1, class T2, class T3>

class Cont{

public:

Cont(int size);

void push(std::shared\_ptr<T1> && t, int index);

void push(std::shared\_ptr<T2> && t, int index);

void push(std::shared\_ptr<T3> && t, int index);

bool IsT1(int index);

bool IsT2(int index);

bool IsT3(int index);

std::shared\_ptr<T1> GetT1(int index);

std::shared\_ptr<T2> GetT2(int index);

std::shared\_ptr<T3> GetT3(int index);

void removeIt(int index);

Iterator<T1, T2, T3> begin();

Iterator<T1, T2, T3> end();

template <class A1, class A2, class A3> friend std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont);

virtual ~Cont();

private:

ContItem<T1, T2, T3> \*a;

int size\_;

Iterator<T1, T2, T3> first;

Iterator<T1, T2, T3> last;

};

#endif

**ContItem.h**

#ifndef CONTITEM\_H

#define CONTITEM\_H

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

template <class T1, class T2, class T3>

class ContItem{

public:

ContItem();

ContItem(int index);

ContItem(std::shared\_ptr<T1> &t, int index);

ContItem(std::shared\_ptr<T2> &t, int index);

ContItem(std::shared\_ptr<T3> &t, int index);

bool IsT1() const;

bool IsT2() const;

bool IsT3() const;

std::shared\_ptr<T1> GetT1();

std::shared\_ptr<T2> GetT2();

std::shared\_ptr<T3> GetT3();

void SetIndex(int index);

int GetIndex();

template <class A1, class A2, class A3> friend std::ostream& operator<< (std::ostream & os, const ContItem<A1, A2, A3> & item);

virtual ~ContItem();

private:

std::shared\_ptr<T1> t1;

std::shared\_ptr<T2> t2;

std::shared\_ptr<T3> t3;

int index\_;

};

#endif

**Iterator.h**

#ifndef ITERATOR\_H

#define ITERATOR\_H

#include <memory>

#include <iostream>

#include "ContItem.cpp"

template<class T1, class T2, class T3>

class Iterator

{

public:

Iterator();

Iterator(ContItem<T1, T2, T3>\* p);

ContItem<T1, T2, T3>\* operator \*();

bool operator==(Iterator i);

bool operator != (Iterator i);

void operator ++ ();

private:

ContItem<T1, T2, T3> \* el;

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0) obj.err=true;

return is;

}

Rectangle::~Rectangle(){

//std::cout << "Прямоугольник удален" << std::endl;

}

**Rhombus.cpp**

#include "Rhombus.h"

#include <iostream>

Rhombus::Rhombus(){

side=0.0;

height=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона = "<<obj.side<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону: "<<std::endl;

if(!(is>>obj.side)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side<0 || obj.height<0) obj.err=true;

return is;

}

Rhombus::~Rhombus(){

//std::cout << "Ромб удален" << std::endl;

}

**Trapezium.cpp**

#include "Trapezium.h"

#include <iostream>

Trapezium::Trapezium(){

side\_a=0.0;

side\_b=0.0;

height=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0 || obj.height<0) obj.err=true;

return is;

}

Trapezium::~Trapezium(){

//std::cout << "Трапеция удален" << std::endl;

}

**Cont.cpp**

#include "Cont.h"

#include <iostream>

template <class T1, class T2, class T3> Cont<T1, T2, T3>::Cont(int size){

a = new ContItem<T1, T2, T3>[size];

size\_ = size;

for (int i = 0; i < size; i++) a[i].SetIndex(i);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T1> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T2> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T3> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT1(int index){

return a[index].IsT1();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT2(int index){

return a[index].IsT2();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT3(int index){

return a[index].IsT3();

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> Cont<T1, T2, T3>::GetT1(int index){

return a[index].GetT1();

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> Cont<T1, T2, T3>::GetT2(int index){

return a[index].GetT2();

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> Cont<T1, T2, T3>::GetT3(int index){

return a[index].GetT3();

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::removeIt(int index){

a[index] = ContItem<T1, T2, T3>();

}

template <class A1, class A2, class A3> std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont){

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

os << "[" << i << "] - " << cont.a[i] << std::endl;

}

return os;

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::begin()

{

return Iterator<T1, T2, T3>(a);

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::end()

{

return Iterator<T1, T2, T3>(&a[size\_]);

}

template <class T1, class T2, class T3> Cont<T1, T2, T3>::~Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem<T1, T2, T3>();

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class Cont<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const Cont<Rectangle, Rhombus, Trapezium> & cont);

**ContItem.cpp**

#include "ContItem.h"

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem() : t1(nullptr), t2(nullptr), t3(nullptr) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(int index) : t1(nullptr), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T1> &t, int index) : t1(t), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T2> &t, int index) : t1(nullptr), t2(t), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T3> &t, int index) : t1(nullptr), t2(nullptr), t3(t), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::~ContItem() {}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT1() const

{

if (t1 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT2() const

{

if (t2 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT3() const

{

if (t3 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> ContItem<T1, T2, T3>::GetT1()

{

return this->t1;

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> ContItem<T1, T2, T3>::GetT2()

{

return this->t2;

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> ContItem<T1, T2, T3>::GetT3()

{

return this->t3;

}

template <class T1, class T2, class T3> void ContItem<T1, T2, T3>::SetIndex(int index)

{

index\_ = index;

}

template <class T1, class T2, class T3> int ContItem<T1, T2, T3>::GetIndex()

{

return index\_;

}

template <class A1, class A2, class A3> std::ostream& operator << (std::ostream &os, const ContItem<A1, A2, A3> &item)

{

if (item.IsT1())

os << \*item.t1 << " (прямоугольник)";

else if (item.IsT2())

os << \*item.t2 << " (ромб)";

else if (item.IsT3())

os << \*item.t3 << " (трапеция)";

else

os << "Пусто";

return os;

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class ContItem<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const ContItem<Rectangle, Rhombus, Trapezium> & item);

**Iterator.cpp**

#include "Iterator.h"

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator()

{

el = nullptr;

}

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator(ContItem<T1, T2, T3>\* p)

{

el = p;

}

template<class T1, class T2, class T3>

ContItem<T1, T2, T3>\* Iterator<T1, T2, T3>::operator \* ()

{

return el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator ==(Iterator i)

{

return el == i.el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator !=(Iterator i)

{

return !(\*this == i);

}

template<class T1, class T2, class T3>

void Iterator<T1, T2, T3>::operator ++ ()

{

el++;

}

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

#include "ContItem.h"

#include "Cont.h"

int main()

{

setlocale(LC\_CTYPE,"rus");

Cont<Rectangle, Rhombus, Trapezium> container(10);

int a;

do{

std::cout<<"1) Добавить фигуру\n2) Вывести содержимое\n3) Удалить фигуру\n4) Вывести фигуру\n5) Выход\n"<<std::endl;

if(!(std::cin>>a)){

std::cin.clear();

while(std::cin.get() !='\n');

}

switch(a){

case 1:{

std::cout << "1) Прямоугольник\n2) Ромб\n3) Трапеция\n" << std::endl;

int v,i;

std::cin >> v;

std::cout << "Введите индекс ";

std::cin >> i;

switch(v){

case 1:

container.push(std::shared\_ptr<Rectangle>(new Rectangle(std::cin)), i);

break;

case 2:

container.push(std::shared\_ptr<Rhombus>(new Rhombus(std::cin)), i);

break;

case 3:

container.push(std::shared\_ptr<Trapezium>(new Trapezium(std::cin)), i);

break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

break;}

case 2:

for (auto i : container) std::cout << \*i << std::endl;

break;

case 3:

std::cout << "Введите индекс ";

int ind;

std::cin >> ind;

container.removeIt(ind);

break;

case 4:

std::cout << "Введите индекс ";

int i;

std::cin >> i;

if (container.IsT1(i))

std::cout << \*container.GetT1(i) << std::endl;

else if (container.IsT2(i))

std::cout << \*container.GetT2(i) << std::endl;

else if (container.IsT3(i))

std::cout << \*container.GetT3(i) << std::endl;

else

std::cout << "Пустой элемент" << std::endl;

break;

case 5:break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

}while(a!=5);

return 0;

}

**Вывод:**

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 1

Введите сторону a:

3

Введите сторону b:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

2

Введите индекс 3

Введите сторону:

2

Введите высоту:

1

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

3

Введите индекс 0

Введите сторону a:

5

Введите сторону b:

3

Введите высоту:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

2

[0] - Сторона а= 5, Сторона b= 3, Высота = 2 (трапеция)

[1] - Сторона а= 3, Сторона b= 2 (прямоугольник)

[2] - Пусто

[3] - Сторона = 2, Высота = 1 (ромб)

[4] - Пусто

[5] - Пусто

[6] - Пусто

[7] - Пусто

[8] - Пусто

[9] - Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

3

Введите индекс 1

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

2

[0] - Сторона а= 5, Сторона b= 3, Высота = 2 (трапеция)

[1] - Пусто

[2] - Пусто

[3] - Сторона = 2, Высота = 1 (ромб)

[4] - Пусто

[5] - Пусто

[6] - Пусто

[7] - Пусто

[8] - Пусто

[9] - Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

4

Введите индекс 0

Сторона а= 5, Сторона b= 3, Высота = 2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

5

Process returned 0 (0x0) execution time : 40.879 s

Press any key to continue.

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.

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

bool err=false;

virtual ~Figure() {}

};

#endif

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure{

public:

Rectangle();

Rectangle(std::istream& is);

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

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

virtual ~Rectangle();

private:

double side\_a;

double side\_b;

};

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(std::istream& is);

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

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

virtual ~Rhombus();

private:

double side;

double height;

};

#endif

**Trapezium.h**

#ifndef TRAPEZIUM\_H

#define TRAPEZIUM\_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure{

public:

Trapezium();

Trapezium(std::istream& is);

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

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

virtual ~Trapezium();

private:

double side\_a;

double side\_b;

double height;

};

#endif

**Cont.h**

#ifndef CONT.H

#define CONT.H

#include <memory>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "ContItem.h"

#include "Iterator.cpp"

const int size = 10;

template <class T1, class T2, class T3>

class Cont{

public:

Cont(int size);

void push(std::shared\_ptr<T1> && t, int index);

void push(std::shared\_ptr<T2> && t, int index);

void push(std::shared\_ptr<T3> && t, int index);

bool IsT1(int index);

bool IsT2(int index);

bool IsT3(int index);

std::shared\_ptr<T1> GetT1(int index);

std::shared\_ptr<T2> GetT2(int index);

std::shared\_ptr<T3> GetT3(int index);

void removeIt(int index);

Iterator<T1, T2, T3> begin();

Iterator<T1, T2, T3> end();

template <class A1, class A2, class A3> friend std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont);

virtual ~Cont();

private:

ContItem<T1, T2, T3> \*a;

int size\_;

Iterator<T1, T2, T3> first;

Iterator<T1, T2, T3> last;

};

#endif

**ContItem.h**

#ifndef CONTITEM\_H

#define CONTITEM\_H

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "Allocator.h"

#include <memory>

#include <iostream>

template <class T1, class T2, class T3>

class ContItem{

public:

ContItem();

ContItem(int index);

ContItem(std::shared\_ptr<T1> &t, int index);

ContItem(std::shared\_ptr<T2> &t, int index);

ContItem(std::shared\_ptr<T3> &t, int index);

bool IsT1() const;

bool IsT2() const;

bool IsT3() const;

std::shared\_ptr<T1> GetT1();

std::shared\_ptr<T2> GetT2();

std::shared\_ptr<T3> GetT3();

void SetIndex(int index);

int GetIndex();

void \*operator new(size\_t size);

void operator delete(void \*ptr);

template <class A1, class A2, class A3> friend std::ostream& operator<< (std::ostream & os, const ContItem<A1, A2, A3> & item);

virtual ~ContItem();

private:

std::shared\_ptr<T1> t1;

std::shared\_ptr<T2> t2;

std::shared\_ptr<T3> t3;

int index\_;

static Allocator arritem\_allocator;

};

#endif

**Iterator.h**

#ifndef ITERATOR\_H

#define ITERATOR\_H

#include <memory>

#include <iostream>

#include "ContItem.cpp"

template<class T1, class T2, class T3>

class Iterator

{

public:

Iterator();

Iterator(ContItem<T1, T2, T3>\* p);

ContItem<T1, T2, T3>\* operator \*();

bool operator==(Iterator i);

bool operator != (Iterator i);

void operator ++ ();

private:

ContItem<T1, T2, T3> \* el;

};

#endif

**Allocator.h**

#ifndef ALLOCATOR\_H

#define ALLOCATOR\_H

#include <cstdlib>

class Allocator {

public:

Allocator(size\_t size,size\_t count);

void \*allocate();

void deallocate(void \*pointer);

bool has\_free\_blocks();

virtual ~Allocator();

private:

size\_t \_size;

size\_t \_count;

char \*\_used\_blocks;

void \*\*\_free\_blocks;

size\_t \_free\_count;

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0) obj.err=true;

return is;

}

Rectangle::~Rectangle(){

//std::cout << "Прямоугольник удален" << std::endl;

}

**Rhombus.cpp**

#include "Rhombus.h"

#include <iostream>

Rhombus::Rhombus(){

side=0.0;

height=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона = "<<obj.side<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону: "<<std::endl;

if(!(is>>obj.side)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side<0 || obj.height<0) obj.err=true;

return is;

}

Rhombus::~Rhombus(){

//std::cout << "Ромб удален" << std::endl;

}

**Trapezium.cpp**

#include "Trapezium.h"

#include <iostream>

Trapezium::Trapezium(){

side\_a=0.0;

side\_b=0.0;

height=0.0;

}

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

is>>\*this;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0 || obj.height<0) obj.err=true;

return is;

}

Trapezium::~Trapezium(){

//std::cout << "Трапеция удалена" << std::endl;

}

**Cont.cpp**

#include "Cont.h"

#include <iostream>

template <class T1, class T2, class T3> Cont<T1, T2, T3>::Cont(int size){

a = new ContItem<T1, T2, T3>[size];

size\_ = size;

for (int i = 0; i < size; i++) a[i].SetIndex(i);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T1> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T2> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T3> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT1(int index){

return a[index].IsT1();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT2(int index){

return a[index].IsT2();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT3(int index){

return a[index].IsT3();

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> Cont<T1, T2, T3>::GetT1(int index){

return a[index].GetT1();

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> Cont<T1, T2, T3>::GetT2(int index){

return a[index].GetT2();

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> Cont<T1, T2, T3>::GetT3(int index){

return a[index].GetT3();

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::removeIt(int index){

a[index] = ContItem<T1, T2, T3>();

}

template <class A1, class A2, class A3> std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont){

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

os << "[" << i << "] - " << cont.a[i] << std::endl;

}

return os;

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::begin()

{

return Iterator<T1, T2, T3>(a);

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::end()

{

return Iterator<T1, T2, T3>(&a[size\_]);

}

template <class T1, class T2, class T3> Cont<T1, T2, T3>::~Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem<T1, T2, T3>();

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class Cont<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const Cont<Rectangle, Rhombus, Trapezium> & cont);

**ContItem.cpp**

#include "ContItem.h"

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem() : t1(nullptr), t2(nullptr), t3(nullptr) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(int index) : t1(nullptr), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T1> &t, int index) : t1(t), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T2> &t, int index) : t1(nullptr), t2(t), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T3> &t, int index) : t1(nullptr), t2(nullptr), t3(t), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::~ContItem() {}

template <class T1, class T2, class T3> Allocator ContItem<T1, T2, T3>::arritem\_allocator(sizeof(ContItem<T1, T2, T3>), 10);

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT1() const

{

if (t1 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT2() const

{

if (t2 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT3() const

{

if (t3 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> ContItem<T1, T2, T3>::GetT1()

{

return this->t1;

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> ContItem<T1, T2, T3>::GetT2()

{

return this->t2;

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> ContItem<T1, T2, T3>::GetT3()

{

return this->t3;

}

template <class T1, class T2, class T3> void ContItem<T1, T2, T3>::SetIndex(int index)

{

index\_ = index;

}

template <class T1, class T2, class T3> int ContItem<T1, T2, T3>::GetIndex()

{

return index\_;

}

template <class T1, class T2, class T3> void \* ContItem<T1, T2, T3>::operator new(size\_t size){

return arritem\_allocator.allocate();

}

template <class T1, class T2, class T3> void ContItem<T1, T2, T3>::operator delete(void \*ptr){

arritem\_allocator.deallocate(ptr);

}

template <class A1, class A2, class A3> std::ostream& operator << (std::ostream &os, const ContItem<A1, A2, A3> &item)

{

if (item.IsT1())

os << \*item.t1 << " (прямоугольник)";

else if (item.IsT2())

os << \*item.t2 << " (ромб)";

else if (item.IsT3())

os << \*item.t3 << " (трапеция)";

else

os << "Пусто";

return os;

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class ContItem<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const ContItem<Rectangle, Rhombus, Trapezium> & item);

**Iterator.cpp**

#include "Iterator.h"

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator()

{

el = nullptr;

}

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator(ContItem<T1, T2, T3>\* p)

{

el = p;

}

template<class T1, class T2, class T3>

ContItem<T1, T2, T3>\* Iterator<T1, T2, T3>::operator \* ()

{

return el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator ==(Iterator i)

{

return el == i.el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator !=(Iterator i)

{

return !(\*this == i);

}

template<class T1, class T2, class T3>

void Iterator<T1, T2, T3>::operator ++ ()

{

el++;

}

**Allocator.cpp**

#include "Allocator.h"

#include <iostream>

Allocator::Allocator(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\_count = \_count;

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

}

void \*Allocator::allocate() {

void \*result = nullptr;

if(\_free\_count>0){

result = \_free\_blocks[\_free\_count-1];

\_free\_count--;

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

}

else{

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

}

return result;

}

void Allocator::deallocate(void \*pointer) {

std::cout << "Allocator: Deallocate"<< std::endl;

\_free\_blocks[\_free\_count] = pointer;

\_free\_count ++;

}

bool Allocator::has\_free\_blocks() {

return \_free\_count>0;

}

Allocator::~Allocator() {

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

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

delete \_free\_blocks;

delete \_used\_blocks;

}

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

#include "ContItem.h"

#include "Cont.h"

#include "Allocator.h"

int main()

{

setlocale(LC\_CTYPE,"rus");

Cont<Rectangle, Rhombus, Trapezium> container(10);

int a;

do{

std::cout<<"1) Добавить фигуру\n2) Вывести содержимое\n3) Удалить фигуру\n4) Вывести фигуру\n5) Выход\n"<<std::endl;

if(!(std::cin>>a)){

std::cin.clear();

while(std::cin.get() !='\n');

}

switch(a){

case 1:{

std::cout << "1) Прямоугольник\n2) Ромб\n3) Трапеция\n" << std::endl;

int v,i;

std::cin >> v;

std::cout << "Введите индекс ";

std::cin >> i;

switch(v){

case 1:

container.push(std::shared\_ptr<Rectangle>(new Rectangle(std::cin)), i);

break;

case 2:

container.push(std::shared\_ptr<Rhombus>(new Rhombus(std::cin)), i);

break;

case 3:

container.push(std::shared\_ptr<Trapezium>(new Trapezium(std::cin)), i);

break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

break;}

case 2:

for (auto i : container) std::cout << \*i << std::endl;

break;

case 3:

std::cout << "Введите индекс ";

int ind;

std::cin >> ind;

container.removeIt(ind);

break;

case 4:

std::cout << "Введите индекс ";

int i;

std::cin >> i;

if (container.IsT1(i))

std::cout << \*container.GetT1(i) << std::endl;

else if (container.IsT2(i))

std::cout << \*container.GetT2(i) << std::endl;

else if (container.IsT3(i))

std::cout << \*container.GetT3(i) << std::endl;

else

std::cout << "Пустой элемент" << std::endl;

break;

case 5:break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

}while(a!=5);

return 0;

}

**Вывод:**

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 1

Введите сторону a:

3

Введите сторону b:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

2

Введите индекс 3

Введите сторону:

2

Введите высоту:

1

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

3

Введите индекс 0

Введите сторону a:

5

Введите сторону b:

3

Введите высоту:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

2

[0] - Сторона а= 5, Сторона b= 3, Высота = 2 (трапеция)

[1] - Сторона а= 3, Сторона b= 2 (прямоугольник)

[2] - Пусто

[3] - Сторона = 2, Высота = 1 (ромб)

[4] - Пусто

[5] - Пусто

[6] - Пусто

[7] - Пусто

[8] - Пусто

[9] - Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

3

Введите индекс 1

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

2

[0] - Сторона а= 5, Сторона b= 3, Высота = 2 (трапеция)

[1] - Пусто

[2] - Пусто

[3] - Сторона = 2, Высота = 1 (ромб)

[4] - Пусто

[5] - Пусто

[6] - Пусто

[7] - Пусто

[8] - Пусто

[9] - Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

4

Введите индекс 0

Сторона а= 5, Сторона b= 3, Высота = 2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Выход

5

Process returned 0 (0x0) execution time : 40.879 s

Press any key to continue.

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).

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

bool err=false;

virtual double Area()=0;

virtual ~Figure() {}

};

#endif

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure{

public:

Rectangle();

Rectangle(std::istream& is);

double Area() override;

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

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

virtual ~Rectangle();

private:

double side\_a;

double side\_b;

};

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(std::istream& is);

double Area() override;

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

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

virtual ~Rhombus();

private:

double side;

double height;

};

#endif

**Trapezium.h**

#ifndef TRAPEZIUM\_H

#define TRAPEZIUM\_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure{

public:

Trapezium();

Trapezium(std::istream& is);

double Area() override;

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

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

virtual ~Trapezium();

private:

double side\_a;

double side\_b;

double height;

};

#endif

**Cont.h**

#ifndef CONT.H

#define CONT.H

#include <memory>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "ContItem.h"

#include "Iterator.cpp"

const int size = 10;

template <class T1, class T2, class T3>

class Cont{

public:

Cont(int size);

void push(std::shared\_ptr<T1> && t, int index);

void push(std::shared\_ptr<T2> && t, int index);

void push(std::shared\_ptr<T3> && t, int index);

void removeIt(double area, int index, int number);

void removeIt(int index, int number);

Iterator<T1, T2, T3> begin();

Iterator<T1, T2, T3> end();

template <class A1, class A2, class A3> friend std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont);

friend int main();

virtual ~Cont();

private:

ContItem<T1, T2, T3> \*a;

int size\_;

Iterator<T1, T2, T3> first;

Iterator<T1, T2, T3> last;

};

#endif

**ContItem.h**

#ifndef CONTITEM\_H

#define CONTITEM\_H

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "Allocator.h"

#include "BTree.h"

#include <memory>

#include <iostream>

template <class T1, class T2, class T3> class Cont;

template <class T1, class T2, class T3>

class ContItem{

public:

ContItem();

ContItem(std::shared\_ptr<BTree<T1,T2,T3>> & t, int index);

/\*ContItem(std::shared\_ptr<T1> &t, int index);

ContItem(std::shared\_ptr<T2> &t, int index);

ContItem(std::shared\_ptr<T3> &t, int index);

bool IsT1() const;

bool IsT2() const;

bool IsT3() const;

std::shared\_ptr<T1> GetT1();

std::shared\_ptr<T2> GetT2();

std::shared\_ptr<T3> GetT3(); \*/

void SetIndex(int index);

int GetIndex(int index);

template <class A1, class A2, class A3> friend std::ostream& operator<< (std::ostream & os, const ContItem<A1, A2, A3> & item);

friend Cont<T1,T2,T3>;

virtual ~ContItem();

private:

std::shared\_ptr<BTree<T1,T2,T3>> t;

int index\_;

};

#endif

**Iterator.h**

#ifndef ITERATOR\_H

#define ITERATOR\_H

#include <memory>

#include <iostream>

#include "ContItem.cpp"

template<class T1, class T2, class T3>

class Iterator

{

public:

Iterator();

Iterator(ContItem<T1, T2, T3>\* p);

ContItem<T1, T2, T3>\* operator \*();

bool operator==(Iterator i);

bool operator != (Iterator i);

void operator ++ ();

private:

ContItem<T1, T2, T3> \* el;

};

#endif

**Allocator.h**

#ifndef ALLOCATOR\_H

#define ALLOCATOR\_H

#include <cstdlib>

class Allocator {

public:

Allocator(size\_t size,size\_t count);

void \*allocate();

void deallocate(void \*pointer);

bool has\_free\_blocks();

virtual ~Allocator();

private:

size\_t \_size;

size\_t \_count;

char \*\_used\_blocks;

void \*\*\_free\_blocks;

size\_t \_free\_count;

};

#endif

**BTree.h**

#ifndef BTREE\_H

#define BTREE\_H

#include <memory>

#include <iostream>

#include "BTreeItem.h"

template <class T1, class T2, class T3> class Cont;

template <class T1, class T2, class T3>

class BTree{

public:

BTree();

BTree(std::shared\_ptr<T1> & t);

BTree(std::shared\_ptr<T2> & t);

BTree(std::shared\_ptr<T3> & t);

/\*std::shared\_ptr<BTreeItem<T1,T2,T3>> find(std::shared\_ptr<T1> &obj);

std::shared\_ptr<BTreeItem<T1,T2,T3>> find(std::shared\_ptr<T2> &obj);

std::shared\_ptr<BTreeItem<T1,T2,T3>> find(std::shared\_ptr<T3> &obj); \*/

void remove(double area);

void push(std::shared\_ptr<T1> & obj);

void push(std::shared\_ptr<T2> & obj);

void push(std::shared\_ptr<T3> & obj);

void print();

void print(std::ostream& os);

bool isEmpty();

template <class A1, class A2, class A3> friend std::ostream& operator<<(std::ostream& os, BTree<A1,A2,A3> &tree);

friend Cont<T1,T2,T3>;

virtual ~BTree();

private:

std::shared\_ptr<BTreeItem<T1,T2,T3>> head;

std::shared\_ptr<BTreeItem<T1,T2,T3>> minValueNode(std::shared\_ptr<BTreeItem<T1,T2,T3>> root);

std::shared\_ptr<BTreeItem<T1,T2,T3>> deleteNode(std::shared\_ptr<BTreeItem<T1,T2,T3>> root, double area);

void print\_tree(std::shared\_ptr<BTreeItem<T1,T2,T3>> item, int32\_t a, std::ostream& os);

int size\_;

};

#endif // BTREE\_H

**BTreeItem.h**

#ifndef BTREEITEM\_H

#define BTREEITEM\_H

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "Allocator.h"

#include <memory>

template <class T1, class T2, class T3> class BTree;

template <class T1, class T2, class T3> class Cont;

template <class T1, class T2, class T3>

class BTreeItem{

public:

BTreeItem();

BTreeItem(std::shared\_ptr<T1> & t1);

BTreeItem(std::shared\_ptr<T2> & t2);

BTreeItem(std::shared\_ptr<T3> & t3);

bool IsT1() const;

bool IsT2() const;

bool IsT3() const;

std::shared\_ptr<T1> GetT1();

std::shared\_ptr<T2> GetT2();

std::shared\_ptr<T3> GetT3();

double Area();

void \*operator new(size\_t size);

void operator delete(void \*ptr);

template <class A1, class A2, class A3> friend std::ostream& operator<< (std::ostream & os, const BTreeItem<A1, A2, A3> & item);

friend BTree<T1,T2,T3>;

friend Cont<T1,T2,T3>;

virtual ~BTreeItem();

private:

std::shared\_ptr<BTreeItem<T1,T2,T3>> leftT;

std::shared\_ptr<BTreeItem<T1,T2,T3>> rightT;

std::shared\_ptr<T1> t1;

std::shared\_ptr<T2> t2;

std::shared\_ptr<T3> t3;

static Allocator arritem\_allocator;

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

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

is>>\*this;

}

double Rectangle::Area(){

return side\_a\*side\_b;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0) obj.err=true;

return is;

}

Rectangle::~Rectangle(){

//std::cout << "Прямоугольник удален" << std::endl;

}

**Rhombus.cpp**

#include "Rhombus.h"

#include <iostream>

Rhombus::Rhombus(){

side=0.0;

height=0.0;

}

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

is>>\*this;

}

double Rhombus::Area(){

return side\*height;

}

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

os<<"Сторона = "<<obj.side<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону: "<<std::endl;

if(!(is>>obj.side)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side<0 || obj.height<0) obj.err=true;

return is;

}

Rhombus::~Rhombus(){

//std::cout << "Ромб удален" << std::endl;

}

**Trapezium.cpp**

#include "Trapezium.h"

#include <iostream>

Trapezium::Trapezium(){

side\_a=0.0;

side\_b=0.0;

height=0.0;

}

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

is>>\*this;

}

double Trapezium::Area(){

return (side\_a+side\_b)/2.0\*height;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0 || obj.height<0) obj.err=true;

return is;

}

Trapezium::~Trapezium(){

//std::cout << "Трапеция удален" << std::endl;

}

**Cont.cpp**

#include "Cont.h"

#include <iostream>

template <class T1, class T2, class T3> Cont<T1, T2, T3>::Cont(int size){

a = new ContItem<T1, T2, T3>[size];

size\_ = size;

for (int i = 0; i < size; i++) a[i].SetIndex(i);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T1> && t, int index){

std::shared\_ptr<BTree<T1,T2,T3>> item = std::make\_shared<BTree<T1,T2,T3>>(t);

a[index]=ContItem<T1, T2, T3>(item, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T2> && t, int index){

std::shared\_ptr<BTree<T1,T2,T3>> item = std::make\_shared<BTree<T1,T2,T3>>(t);

a[index]=ContItem<T1, T2, T3>(item, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T3> && t, int index){

std::shared\_ptr<BTree<T1,T2,T3>> item = std::make\_shared<BTree<T1,T2,T3>>(t);

a[index]=ContItem<T1, T2, T3>(item, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::removeIt(double area, int index,int number){

if(number == 1){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = a[index].t->head;

for(int i=1; i<6; i++){

if(item->Area() < area){

a[index].t->remove(item->Area());

a[index].t->size\_--;

item = a[index].t->head;

continue;

}

else if(item->leftT){

item = item->leftT;

continue;

}

else if(item->rightT){

item = item->rightT;

continue;

}

}

return;

}

else if(number == 2){

a[index].t->remove(area);

a[index].t->size\_--;

return;

}

else if(number == 3){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = a[index].t->head;

for(int i=1; i<6; i++){

if(item->Area() > area){

a[index].t->remove(item->Area());

a[index].t->size\_--;

item = a[index].t->head;

continue;

}

else if(item->rightT){

item = item->rightT;

continue;

}

else if(item->leftT){

item = item->leftT;

continue;

}

}

return;

}

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::removeIt(int index,int number){

if(number == 1){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = a[index].t->head;

for(int i=1; i<6; i++){

if(item->IsT1()){

a[index].t->remove(item->Area());

a[index].t->size\_--;

item = a[index].t->head;

continue;

}

else if(item->leftT){

item = item->leftT;

continue;

}

else if(item->rightT){

item = item->rightT;

continue;

}

}

return;

}

else if(number == 2){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = a[index].t->head;

for(int i=1; i<6; i++){

if(item->IsT2()){

a[index].t->remove(item->Area());

a[index].t->size\_--;

item = a[index].t->head;

continue;

}

else if(item->leftT){

item = item->leftT;

continue;

}

else if(item->rightT){

item = item->rightT;

continue;

}

}

return;

}

else if(number == 3){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = a[index].t->head;

for(int i=1; i<6; i++){

if(item->IsT3()){

a[index].t->remove(item->Area());

a[index].t->size\_--;

item = a[index].t->head;

continue;

}

else if(item->leftT){

item = item->leftT;

continue;

}

else if(item->rightT){

item = item->rightT;

continue;

}

}

return;

}

}

template <class A1, class A2, class A3> std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont){

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

os << "[" << i << "] - " << cont.a[i] << std::endl;

}

return os;

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::begin()

{

return Iterator<T1, T2, T3>(a);

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::end()

{

return Iterator<T1, T2, T3>(&a[size\_]);

}

template <class T1, class T2, class T3> Cont<T1, T2, T3>::~Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem<T1, T2, T3>();

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class Cont<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const Cont<Rectangle, Rhombus, Trapezium> & cont);

**ContItem.cpp**

#include "ContItem.h"

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem() : t(nullptr) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<BTree<T1,T2,T3>> & t, int index) : t(t), index\_(index) {}

/\*template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T1> &t, int index) : t1(t), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T2> &t, int index) : t1(nullptr), t2(t), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T3> &t, int index) : t1(nullptr), t2(nullptr), t3(t), index\_(index) {}

\*/

/\*template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT1() const

{

if (t1 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT2() const

{

if (t2 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT3() const

{

if (t3 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> ContItem<T1, T2, T3>::GetT1()

{

return this->t1;

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> ContItem<T1, T2, T3>::GetT2()

{

return this->t2;

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> ContItem<T1, T2, T3>::GetT3()

{

return this->t3;

} \*/

template <class T1, class T2, class T3> void ContItem<T1, T2, T3>::SetIndex(int index)

{

index\_ = index;

return;

}

template <class T1, class T2, class T3> int ContItem<T1, T2, T3>::GetIndex(int index)

{

return index\_;

}

template <class A1, class A2, class A3> std::ostream& operator << (std::ostream &os, const ContItem<A1, A2, A3> &item)

{

if (item.t)

os << \*item.t << std::endl;

else

os << "Пусто" << std::endl;

return os;

}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::~ContItem() {

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class ContItem<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const ContItem<Rectangle, Rhombus, Trapezium> & item);

**Iterator.cpp**

#include "Iterator.h"

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator()

{

el = nullptr;

}

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator(ContItem<T1, T2, T3>\* p)

{

el = p;

}

template<class T1, class T2, class T3>

ContItem<T1, T2, T3>\* Iterator<T1, T2, T3>::operator \* ()

{

return el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator ==(Iterator i)

{

return el == i.el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator !=(Iterator i)

{

return !(\*this == i);

}

template<class T1, class T2, class T3>

void Iterator<T1, T2, T3>::operator ++ ()

{

el++;

}

**Allocator.cpp**

#include "Allocator.h"

#include <iostream>

Allocator::Allocator(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\_count = \_count;

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

}

void \*Allocator::allocate() {

void \*result = nullptr;

if(\_free\_count>0){

result = \_free\_blocks[\_free\_count-1];

\_free\_count--;

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

}

else{

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

}

return result;

}

void Allocator::deallocate(void \*pointer) {

std::cout << "Allocator: Deallocate"<< std::endl;

\_free\_blocks[\_free\_count] = pointer;

\_free\_count ++;

}

bool Allocator::has\_free\_blocks() {

return \_free\_count>0;

}

Allocator::~Allocator() {

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

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

delete \_free\_blocks;

delete \_used\_blocks;

}

**BTree.cpp**

#include "BTree.h"

template <class T1, class T2, class T3> BTree<T1,T2,T3>::BTree(){

head = nullptr;

size\_ = 0;

}

template <class T1, class T2, class T3> BTree<T1,T2,T3>::BTree(std::shared\_ptr<T1> & t){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = std::make\_shared<BTreeItem<T1,T2,T3>>(t);

head = item;

size\_ = 1;

}

template <class T1, class T2, class T3> BTree<T1,T2,T3>::BTree(std::shared\_ptr<T2> & t){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = std::make\_shared<BTreeItem<T1,T2,T3>>(t);

head = item;

size\_ = 1;

}

template <class T1, class T2, class T3> BTree<T1,T2,T3>::BTree(std::shared\_ptr<T3> & t){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = std::make\_shared<BTreeItem<T1,T2,T3>>(t);

head = item;

size\_ = 1;

}

template <class T1, class T2, class T3> std::shared\_ptr<BTreeItem<T1,T2,T3>> BTree<T1,T2,T3>::minValueNode(std::shared\_ptr<BTreeItem<T1,T2,T3>> root){

std::shared\_ptr<BTreeItem<T1,T2,T3>> min = root;

while (min->leftT != nullptr) {

min = min->leftT;

}

return min;

}

/\*template <class T1, class T2, class T3> std::shared\_ptr<BTreeItem<T1,T2,T3>> BTree<T1,T2,T3>::find(std::shared\_ptr<T1> &obj){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = head;

while (item != nullptr) {

if (item->Area() == obj->Area()) {

return item;

}

else if (item->Area() > obj->Area()) {

item = item->leftT;

}

else if (item->Area() < obj->Area()) {

item = item->rightT;

}

}

return nullptr;

}

template <class T1, class T2, class T3> std::shared\_ptr<BTreeItem<T1,T2,T3>> BTree<T1,T2,T3>::find(std::shared\_ptr<T2> &obj){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = head;

while (item != nullptr) {

if (item->Area() == obj->Area()) {

return item;

}

else if (item->Area() > obj->Area()) {

item = item->leftT;

}

else if (item->Area() < obj->Area()) {

item = item->rightT;

}

}

return nullptr;

}

template <class T1, class T2, class T3> std::shared\_ptr<BTreeItem<T1,T2,T3>> BTree<T1,T2,T3>::find(std::shared\_ptr<T3> &obj){

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = head;

while (item != nullptr) {

if (item->Area() == obj->Area()) {

return item;

}

else if (item->Area() > obj->Area()) {

item = item->leftT;

}

else if (item->Area() < obj->Area()) {

item = item->rightT;

}

}

return nullptr;

} \*/

template <class T1, class T2, class T3> std::shared\_ptr<BTreeItem<T1,T2,T3>> BTree<T1,T2,T3>::deleteNode(std::shared\_ptr<BTreeItem<T1,T2,T3>> root, double area){

if (root == nullptr) {

return root;

}

if (area < root->Area()) {

root->leftT = deleteNode(root->leftT, area);

}

else if (area > root->Area()) {

root->rightT = deleteNode(root->rightT, area);

}

else {

if (root->leftT == nullptr) {

std::shared\_ptr<BTreeItem<T1,T2,T3>> temp = root->rightT;

root->leftT = nullptr;

root->rightT = nullptr;

return temp;

}

else if (root->rightT == nullptr) {

std::shared\_ptr<BTreeItem<T1,T2,T3>> temp = root->leftT;

root->leftT = nullptr;

root->rightT = nullptr;

return temp;

}

std::shared\_ptr<BTreeItem<T1,T2,T3>> temp = minValueNode(root->rightT);

root->t1 = temp->t1;

root->t2 = temp->t2;

root->t3 = temp->t3;

root->rightT = deleteNode(root->rightT, temp->Area());

}

return root;

}

template <class T1, class T2, class T3> void BTree<T1,T2,T3>::remove(double area){

head = BTree<T1,T2,T3>::deleteNode(head, area);

}

template <class T1, class T2, class T3> void BTree<T1,T2,T3>::push(std::shared\_ptr<T1> & obj){

if(size\_>5){

std::cout << "Добавление невозможно, выберете дркгой эл-т массива" << std::endl;

return;

}

if (head == nullptr) {

head = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

size\_++;

return;

}

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = head;

while (true) {

if (obj->Area() <= item->Area()) {

if (item->leftT == nullptr) {

item->leftT = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

break;

}

else {

item = item->leftT;

}

}

else {

if (item->rightT == nullptr) {

item->rightT = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

break;

}

else {

item = item->rightT;

}

}

}

}

template <class T1, class T2, class T3> void BTree<T1,T2,T3>::push(std::shared\_ptr<T2> & obj){

if(size\_>5){

std::cout << "Добавление невозможно, выберете дркгой эл-т массива" << std::endl;

return;

}

if (head == nullptr) {

head = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

size\_++;

return;

}

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = head;

while (true) {

if (obj->Area() <= item->Area()) {

if (item->leftT == nullptr) {

item->leftT = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

break;

}

else {

item = item->leftT;

}

}

else {

if (item->rightT == nullptr) {

item->rightT = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

break;

}

else {

item = item->rightT;

}

}

}

}

template <class T1, class T2, class T3> void BTree<T1,T2,T3>::push(std::shared\_ptr<T3> & obj){

if(size\_>5){

std::cout << "Добавление невозможно, выберете дркгой эл-т массива" << std::endl;

return;

}

if (head == nullptr) {

head = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

size\_++;

return;

}

std::shared\_ptr<BTreeItem<T1,T2,T3>> item = head;

while (true) {

if (obj->Area() <= item->Area()) {

if (item->leftT == nullptr) {

item->leftT = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

break;

}

else {

item = item->leftT;

}

}

else {

if (item->rightT == nullptr) {

item->rightT = std::make\_shared<BTreeItem<T1,T2,T3>>(obj);

break;

}

else {

item = item->rightT;

}

}

}

}

template <class T1, class T2, class T3> void BTree<T1,T2,T3>::print\_tree(std::shared\_ptr<BTreeItem<T1,T2,T3>> item, int32\_t a, std::ostream& os){

for (int32\_t i = 0; i < a; i++) {

os << " ";

}

os << \*item; //item->Area(); //item->GetFigure()->Print();

if (item->leftT != nullptr) {

BTree<T1,T2,T3>::print\_tree(item->leftT, a + 1, os);

}

else if (item->rightT != nullptr) {

for (int32\_t i = 0; i <= a; i++) {

os << " ";

}

os << "null" << std::endl;

}

if (item->rightT != nullptr) {

BTree<T1,T2,T3>::print\_tree(item->rightT, a + 1, os);

}

else if (item->leftT != nullptr) {

for (int32\_t i = 0; i <= a; i++) {

os << " ";

}

os << "null" << std::endl;

}

}

template <class T1, class T2, class T3> void BTree<T1,T2,T3>::print(){

if (head != nullptr) {

BTree<T1,T2,T3>::print\_tree(head, 0, std::cout);

}

}

template <class T1, class T2, class T3> void BTree<T1,T2,T3>::print(std::ostream& os){

if (head != nullptr) {

BTree<T1,T2,T3>::print\_tree(head, 0, os);

}

}

template <class T1, class T2, class T3> bool BTree<T1,T2,T3>::isEmpty(){

return head == nullptr;

}

template <class A1, class A2, class A3> std::ostream& operator<<(std::ostream& os, BTree<A1,A2,A3>& tree) {

tree.print(os);

return os;

}

template <class T1, class T2, class T3> BTree<T1,T2,T3>::~BTree(){

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class BTree<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os, BTree<Rectangle, Rhombus, Trapezium> & cont);

**BTreeItem.cpp**

#include "BTreeItem.h"

template <class T1, class T2, class T3> BTreeItem<T1,T2,T3>::BTreeItem() : leftT(nullptr), rightT(nullptr), t1(nullptr), t2(nullptr), t3(nullptr) {}

template <class T1, class T2, class T3> BTreeItem<T1,T2,T3>::BTreeItem(std::shared\_ptr<T1> & t1) : leftT(nullptr), rightT(nullptr), t1(t1), t2(nullptr), t3(nullptr) {}

template <class T1, class T2, class T3> BTreeItem<T1,T2,T3>::BTreeItem(std::shared\_ptr<T2> & t2) : leftT(nullptr), rightT(nullptr), t1(nullptr), t2(t2), t3(nullptr) {}

template <class T1, class T2, class T3> BTreeItem<T1,T2,T3>::BTreeItem(std::shared\_ptr<T3> & t3) : leftT(nullptr), rightT(nullptr), t1(nullptr), t2(nullptr), t3(t3) {}

template <class T1, class T2, class T3> Allocator BTreeItem<T1, T2, T3>::arritem\_allocator(sizeof(BTreeItem<T1, T2, T3>), 5);

template <class T1, class T2, class T3> bool BTreeItem<T1,T2,T3>::IsT1() const{

if (t1 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool BTreeItem<T1,T2,T3>::IsT2() const{

if (t2 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool BTreeItem<T1,T2,T3>::IsT3() const{

if (t3 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> BTreeItem<T1, T2, T3>::GetT1()

{

return this->t1;

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> BTreeItem<T1, T2, T3>::GetT2()

{

return this->t2;

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> BTreeItem<T1, T2, T3>::GetT3()

{

return this->t3;

}

template <class T1, class T2, class T3> double BTreeItem<T1, T2, T3>::Area()

{

if(t1) return t1->Area();

else if(t2) return t2->Area();

else if(t3) return t3->Area();

return 0;

}

template <class T1, class T2, class T3> void \* BTreeItem<T1, T2, T3>::operator new(size\_t size){

return arritem\_allocator.allocate();

}

template <class T1, class T2, class T3> void BTreeItem<T1, T2, T3>::operator delete(void \*ptr){

arritem\_allocator.deallocate(ptr);

}

template <class A1, class A2, class A3> std::ostream& operator << (std::ostream &os, const BTreeItem<A1, A2, A3> & item){

if (item.IsT1())

os << \*item.t1 << " (прямоугольник)";

else if (item.IsT2())

os << \*item.t2 << " (ромб)";

else if (item.IsT3())

os << \*item.t3 << " (трапеция)";

else

os << "Пусто";

return os;

}

template <class T1, class T2, class T3> BTreeItem<T1, T2, T3>::~BTreeItem(){

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class BTreeItem<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const BTreeItem<Rectangle, Rhombus, Trapezium> & item);

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

#include "ContItem.h"

#include "Cont.h"

#include "BTree.h"

#include "Allocator.h"

int main()

{

setlocale(LC\_CTYPE,"rus");

Cont<Rectangle, Rhombus, Trapezium> container(10);

int a;

do{

std::cout<<"1) Добавить фигуру\n2) Вывести содержимое\n3) Удалить фигуру\n4) Выход\n"<<std::endl;

if(!(std::cin>>a)){

std::cin.clear();

while(std::cin.get() !='\n');

}

switch(a){

case 1:{

std::cout << "1) Прямоугольник\n2) Ромб\n3) Трапеция\n" << std::endl;

int v,i;

std::cin >> v;

std::cout << "Введите индекс ";

std::cin >> i;

switch(v){

case 1:

container.push(std::shared\_ptr<Rectangle>(new Rectangle(std::cin)), i);

break;

case 2:

container.push(std::shared\_ptr<Rhombus>(new Rhombus(std::cin)), i);

break;

case 3:

container.push(std::shared\_ptr<Trapezium>(new Trapezium(std::cin)), i);

break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

break;}

case 2:

int ind\_1;

std::cout << "1) 1-ый контейнер\n2) 2-ой контейнер\n" << std::endl;

std::cin >> ind\_1;

switch(ind\_1){

case 1:

for (auto i : container) std::cout << \*i << std::endl;

break;

case 2:

int ind\_2;

std::cout << "Введите индекс ";

std::cin >> ind\_2;

std::cout << container.a[ind\_2] << std::endl;

break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

break;

case 3:{

std::cout << "Введите индекс: ";

int ind\_1,ind;

std::cin >> ind;

std::cout << "1) по типу фигуры\n2) по площади\n" << std::endl;

std::cin >> ind\_1;

switch(ind\_1){

case 1:{

int ind\_2;

std::cout << "1) Прямоугольник\n2) Ромб\n3) Трапеция\n" << std::endl;

std::cin >> ind\_2;

container.removeIt(ind,ind\_2);

break; }

case 2:{

int ind\_2;

double area;

std::cout << "1) < значения\n2) == значению\n3) > значения\n" << std::endl;

std::cin >> ind\_2;

std::cout << "Введите площадь: " << std::endl;

std::cin >> area;

container.removeIt(area,ind,ind\_2);

break; }

default:{

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break; }

}

break;

/\*case 4:

std::cout << "Введите индекс ";

int i;

std::cin >> i;

if (container.IsT1(i))

std::cout << \*container.GetT1(i) << std::endl;

else if (container.IsT2(i))

std::cout << \*container.GetT2(i) << std::endl;

else if (container.IsT3(i))

std::cout << \*container.GetT3(i) << std::endl;

else

std::cout << "Пустой элемент" << std::endl;

break; \*/

case 4:break;

default:{

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break; }

}

}}while(a!=4);

return 0;

}

**Вывод:**

Allocator: Memory init

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 1

Введите сторону a:

1

Введите сторону b:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 3

Введите сторону a:

4

Введите сторону b:

3

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

2

1) 1-ый контейнер

2) 2-ой контейнер

1

Пусто

Сторона а= 1, Сторона b= 2 (прямоугольник)

Пусто

Сторона а= 4, Сторона b= 3 (прямоугольник)

Пусто

Пусто

Пусто

Пусто

Пусто

Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Выход

2

1) 1-ый контейнер

2) 2-ой контейнер

2

Введите индекс 3

Сторона а= 4, Сторона b= 3 (прямоугольник)

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

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

bool err=false;

virtual double Area()=0;

virtual ~Figure() {}

};

#endif

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure{

public:

Rectangle();

Rectangle(std::istream& is);

double Area() override;

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

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

virtual ~Rectangle();

private:

double side\_a;

double side\_b;

};

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(std::istream& is);

double Area() override;

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

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

virtual ~Rhombus();

private:

double side;

double height;

};

#endif

**Trapezium.h**

#ifndef TRAPEZIUM\_H

#define TRAPEZIUM\_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure{

public:

Trapezium();

Trapezium(std::istream& is);

double Area() override;

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

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

virtual ~Trapezium();

private:

double side\_a;

double side\_b;

double height;

};

#endif

**Cont.h**

#ifndef CONT.H

#define CONT.H

#include <memory>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "ContItem.h"

#include "Iterator.cpp"

#include <future>

#include <mutex>

const int size = 10;

template <class T1, class T2, class T3>

class Cont{

public:

Cont(int size);

void push(std::shared\_ptr<T1> && t, int index);

void push(std::shared\_ptr<T2> && t, int index);

void push(std::shared\_ptr<T3> && t, int index);

bool IsT1(int index);

bool IsT2(int index);

bool IsT3(int index);

std::shared\_ptr<T1> GetT1(int index);

std::shared\_ptr<T2> GetT2(int index);

std::shared\_ptr<T3> GetT3(int index);

void removeIt(int index);

void QuickSort(int L, int R);

void QuickSortParallel(int L, int R);

Iterator<T1, T2, T3> begin();

Iterator<T1, T2, T3> end();

template <class A1, class A2, class A3> friend std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont);

virtual ~Cont();

private:

ContItem<T1, T2, T3> \*a;

int size\_;

Iterator<T1, T2, T3> first;

Iterator<T1, T2, T3> last;

std::future<void> QuickSortInBackground(int L, int R);

};

#endif

**ContItem.h**

#ifndef CONTITEM\_H

#define CONTITEM\_H

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "Allocator.h"

#include <memory>

#include <iostream>

template <class T1, class T2, class T3>

class ContItem{

public:

ContItem();

ContItem(int index);

ContItem(std::shared\_ptr<T1> &t, int index);

ContItem(std::shared\_ptr<T2> &t, int index);

ContItem(std::shared\_ptr<T3> &t, int index);

bool IsT1() const;

bool IsT2() const;

bool IsT3() const;

std::shared\_ptr<T1> GetT1();

std::shared\_ptr<T2> GetT2();

std::shared\_ptr<T3> GetT3();

void SetIndex(int index);

int GetIndex();

double Area\_();

void \*operator new(size\_t size);

void operator delete(void \*ptr);

template <class A1, class A2, class A3> friend std::ostream& operator<< (std::ostream & os, const ContItem<A1, A2, A3> & item);

virtual ~ContItem();

private:

std::shared\_ptr<T1> t1;

std::shared\_ptr<T2> t2;

std::shared\_ptr<T3> t3;

int index\_;

static Allocator arritem\_allocator;

};

#endif

**Iterator.h**

#ifndef ITERATOR\_H

#define ITERATOR\_H

#include <memory>

#include <iostream>

#include "ContItem.cpp"

template<class T1, class T2, class T3>

class Iterator

{

public:

Iterator();

Iterator(ContItem<T1, T2, T3>\* p);

ContItem<T1, T2, T3>\* operator \*();

bool operator==(Iterator i);

bool operator != (Iterator i);

void operator ++ ();

private:

ContItem<T1, T2, T3> \* el;

};

#endif

**Allocator.h**

#ifndef ALLOCATOR\_H

#define ALLOCATOR\_H

#include <cstdlib>

class Allocator {

public:

Allocator(size\_t size,size\_t count);

void \*allocate();

void deallocate(void \*pointer);

bool has\_free\_blocks();

virtual ~Allocator();

private:

size\_t \_size;

size\_t \_count;

char \*\_used\_blocks;

void \*\*\_free\_blocks;

size\_t \_free\_count;

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

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

is>>\*this;

}

double Rectangle::Area(){

return side\_a\*side\_b;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0) obj.err=true;

return is;

}

Rectangle::~Rectangle(){

//std::cout << "Прямоугольник удален" << std::endl;

}

**Rhombus.cpp**

#include "Rhombus.h"

#include <iostream>

Rhombus::Rhombus(){

side=0.0;

height=0.0;

}

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

is>>\*this;

}

double Rhombus::Area(){

return side\*height;

}

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

os<<"Сторона = "<<obj.side<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону: "<<std::endl;

if(!(is>>obj.side)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side<0 || obj.height<0) obj.err=true;

return is;

}

Rhombus::~Rhombus(){

//std::cout << "Ромб удален" << std::endl;

}

**Trapezium.cpp**

#include "Trapezium.h"

#include <iostream>

Trapezium::Trapezium(){

side\_a=0.0;

side\_b=0.0;

height=0.0;

}

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

is>>\*this;

}

double Trapezium::Area(){

return (side\_a+side\_b)/2.0\*height;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0 || obj.height<0) obj.err=true;

return is;

}

Trapezium::~Trapezium(){

//std::cout << "Трапеция удален" << std::endl;

}

**Cont.cpp**

#include "Cont.h"

#include <iostream>

#include <algorithm>

template <class T1, class T2, class T3> Cont<T1, T2, T3>::Cont(int size){

a = new ContItem<T1, T2, T3>[size];

size\_ = size;

for (int i = 0; i < size; i++) a[i].SetIndex(i);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T1> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T2> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T3> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT1(int index){

return a[index].IsT1();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT2(int index){

return a[index].IsT2();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT3(int index){

return a[index].IsT3();

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> Cont<T1, T2, T3>::GetT1(int index){

return a[index].GetT1();

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> Cont<T1, T2, T3>::GetT2(int index){

return a[index].GetT2();

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> Cont<T1, T2, T3>::GetT3(int index){

return a[index].GetT3();

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::removeIt(int index){

a[index] = ContItem<T1, T2, T3>();

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::QuickSort(int L, int R){

int left = L;

int right = R;

double middle = a[(right + left) / 2].Area\_();

do

{

while (a[left].Area\_() < middle) left++;

while (a[right].Area\_() > middle) right--;

if (left <= right)

{

if (a[left].Area\_() > a[right].Area\_()) std::swap(a[left], a[right]);

left++;

right--;

}

} while (left <= right);

if (right > L) QuickSort(L, right);

if (left < R) QuickSort(left, R);

}

template <class T1, class T2, class T3>

std::future<void> Cont<T1, T2, T3>::QuickSortInBackground(int L, int R)

{

std::packaged\_task<void(int, int)> task

(

std::bind

(

&Cont<T1, T2, T3>::QuickSortParallel,

this,

std::placeholders::\_1,

std::placeholders::\_2

)

);

std::future<void> res(task.get\_future());

std::thread th(std::move(task), L, R);

th.detach();

return res;

}

template <class T1, class T2, class T3>

void Cont<T1, T2, T3>::QuickSortParallel(int L, int R)

{

int left = L;

int right = R;

double middle = a[(right + left) / 2].Area\_();

do

{

while (a[left].Area\_() < middle) left++;

while (a[right].Area\_() > middle) right--;

if (left <= right)

{

if (a[left].Area\_() > a[right].Area\_()) std::swap(a[left], a[right]);

left++;

right--;

}

} while (left <= right);

std::future<void> left\_res;

std::future<void> right\_res;

if (right > L) left\_res = QuickSortInBackground(L, right);

if (left < R) right\_res = QuickSortInBackground(left, R);

if (right > L) left\_res.get();

if (left < R) right\_res.get();

}

template <class A1, class A2, class A3> std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont){

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

os << "[" << i << "] - " << cont.a[i] << std::endl;

}

return os;

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::begin()

{

return Iterator<T1, T2, T3>(a);

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::end()

{

return Iterator<T1, T2, T3>(&a[size\_]);

}

template <class T1, class T2, class T3> Cont<T1, T2, T3>::~Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem<T1, T2, T3>();

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class Cont<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const Cont<Rectangle, Rhombus, Trapezium> & cont);

**ContItem.cpp**

#include "ContItem.h"

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem() : t1(nullptr), t2(nullptr), t3(nullptr) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(int index) : t1(nullptr), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T1> &t, int index) : t1(t), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T2> &t, int index) : t1(nullptr), t2(t), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T3> &t, int index) : t1(nullptr), t2(nullptr), t3(t), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::~ContItem() {}

template <class T1, class T2, class T3> Allocator ContItem<T1, T2, T3>::arritem\_allocator(sizeof(ContItem<T1, T2, T3>), 10);

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT1() const

{

if (t1 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT2() const

{

if (t2 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT3() const

{

if (t3 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> ContItem<T1, T2, T3>::GetT1()

{

return this->t1;

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> ContItem<T1, T2, T3>::GetT2()

{

return this->t2;

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> ContItem<T1, T2, T3>::GetT3()

{

return this->t3;

}

template <class T1, class T2, class T3> void ContItem<T1, T2, T3>::SetIndex(int index)

{

index\_ = index;

}

template <class T1, class T2, class T3> int ContItem<T1, T2, T3>::GetIndex()

{

return index\_;

}

template <class T1, class T2, class T3> double ContItem<T1, T2, T3>::Area\_()

{

if(t1) return t1->Area();

else if(t2) return t2->Area();

else if(t3) return t3->Area();

else return 0;

}

template <class T1, class T2, class T3> void \* ContItem<T1, T2, T3>::operator new(size\_t size){

return arritem\_allocator.allocate();

}

template <class T1, class T2, class T3> void ContItem<T1, T2, T3>::operator delete(void \*ptr){

arritem\_allocator.deallocate(ptr);

}

template <class A1, class A2, class A3> std::ostream& operator << (std::ostream &os, const ContItem<A1, A2, A3> &item)

{

if (item.IsT1())

os << \*item.t1 << " (прямоугольник)";

else if (item.IsT2())

os << \*item.t2 << " (ромб)";

else if (item.IsT3())

os << \*item.t3 << " (трапеция)";

else

os << "Пусто";

return os;

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class ContItem<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const ContItem<Rectangle, Rhombus, Trapezium> & item);

**Iterator.cpp**

#include "Iterator.h"

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator()

{

el = nullptr;

}

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator(ContItem<T1, T2, T3>\* p)

{

el = p;

}

template<class T1, class T2, class T3>

ContItem<T1, T2, T3>\* Iterator<T1, T2, T3>::operator \* ()

{

return el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator ==(Iterator i)

{

return el == i.el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator !=(Iterator i)

{

return !(\*this == i);

}

template<class T1, class T2, class T3>

void Iterator<T1, T2, T3>::operator ++ ()

{

el++;

}

**Allocator.cpp**

#include "Allocator.h"

#include <iostream>

Allocator::Allocator(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\_count = \_count;

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

}

void \*Allocator::allocate() {

void \*result = nullptr;

if(\_free\_count>0){

result = \_free\_blocks[\_free\_count-1];

\_free\_count--;

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

}

else{

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

}

return result;

}

void Allocator::deallocate(void \*pointer) {

std::cout << "Allocator: Deallocate"<< std::endl;

\_free\_blocks[\_free\_count] = pointer;

\_free\_count ++;

}

bool Allocator::has\_free\_blocks() {

return \_free\_count>0;

}

Allocator::~Allocator() {

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

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

delete \_free\_blocks;

delete \_used\_blocks;

}

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

#include "ContItem.h"

#include "Cont.h"

#include "Allocator.h"

#include <future>

int main()

{

setlocale(LC\_CTYPE,"rus");

Cont<Rectangle, Rhombus, Trapezium> container(10);

int a;

do{

std::cout<<"1) Добавить фигуру\n2) Вывести содержимое\n3) Удалить фигуру\n4) Вывести фигуру\n5) Сортировка\n6) Параллельная сортировка\n7) Выход\n"<<std::endl;

if(!(std::cin>>a)){

std::cin.clear();

while(std::cin.get() !='\n');

}

switch(a){

case 1:{

std::cout << "1) Прямоугольник\n2) Ромб\n3) Трапеция\n" << std::endl;

int v,i;

std::cin >> v;

std::cout << "Введите индекс ";

std::cin >> i;

switch(v){

case 1:

container.push(std::shared\_ptr<Rectangle>(new Rectangle(std::cin)), i);

break;

case 2:

container.push(std::shared\_ptr<Rhombus>(new Rhombus(std::cin)), i);

break;

case 3:

container.push(std::shared\_ptr<Trapezium>(new Trapezium(std::cin)), i);

break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

break;}

case 2:

for (auto i : container) std::cout << \*i << std::endl;

break;

case 3:

std::cout << "Введите индекс ";

int ind;

std::cin >> ind;

container.removeIt(ind);

break;

case 4:

std::cout << "Введите индекс ";

int i;

std::cin >> i;

if (container.IsT1(i))

std::cout << \*container.GetT1(i) << std::endl;

else if (container.IsT2(i))

std::cout << \*container.GetT2(i) << std::endl;

else if (container.IsT3(i))

std::cout << \*container.GetT3(i) << std::endl;

else

std::cout << "Пустой элемент" << std::endl;

break;

case 5:

container.QuickSort(0,9);

break;

case 6:

container.QuickSortParallel(0,9);

break;

case 7:break;

default:

std::cout<<"Ошибка. Такого пункта меню не существует\n"<<std::endl;

break;

}

}while(a!=7);

return 0;

}

**Вывод:**

Allocator: Memory init

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 1

Введите сторону a:

1

Введите сторону b:

2

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

2

Введите индекс 0

Введите сторону:

6

Введите высоту:

5

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 4

Введите сторону a:

9

Введите сторону b:

9

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 7

Введите сторону a:

2

Введите сторону b:

3

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

2

Введите индекс 5

Введите сторону:

4

Введите высоту:

3

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

2

Сторона = 6, Высота = 5 (ромб)

Сторона а= 1, Сторона b= 2 (прямоугольник)

Пусто

Пусто

Сторона а= 9, Сторона b= 9 (прямоугольник)

Сторона = 4, Высота = 3 (ромб)

Пусто

Сторона а= 2, Сторона b= 3 (прямоугольник)

Пусто

Пусто

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

5

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

2

Пусто

Пусто

Пусто

Пусто

Пусто

Сторона а= 1, Сторона b= 2 (прямоугольник)

Сторона а= 2, Сторона b= 3 (прямоугольник)

Сторона = 4, Высота = 3 (ромб)

Сторона = 6, Высота = 5 (ромб)

Сторона а= 9, Сторона b= 9 (прямоугольник)

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

1

1) Прямоугольник

2) Ромб

3) Трапеция

1

Введите индекс 3

Введите сторону a:

4

Введите сторону b:

4

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

6

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

2

Пусто

Пусто

Пусто

Пусто

Сторона а= 1, Сторона b= 2 (прямоугольник)

Сторона а= 2, Сторона b= 3 (прямоугольник)

Сторона = 4, Высота = 3 (ромб)

Сторона а= 4, Сторона b= 4 (прямоугольник)

Сторона = 6, Высота = 5 (ромб)

Сторона а= 9, Сторона b= 9 (прямоугольник)

1) Добавить фигуру

2) Вывести содержимое

3) Удалить фигуру

4) Вывести фигуру

5) Сортировка

6) Параллельная сортировка

7) Выход

7

Allocator: Memory freed

Process returned 0 (0x0) execution time : 91.511 s

Press any key to continue.

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.

**Figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

class Figure{

public:

bool err=false;

virtual double Area()=0;

virtual ~Figure() {}

};

#endif

**Rectangle.h**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "Figure.h"

class Rectangle : public Figure{

public:

Rectangle();

Rectangle(std::istream& is);

Rectangle(double a,double b);

double Area() override;

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

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

virtual ~Rectangle();

private:

double side\_a;

double side\_b;

};

#endif

**Rhombus.h**

#ifndef RHOMBUS\_H

#define RHOMBUS\_H

#include <iostream>

#include "Figure.h"

class Rhombus : public Figure{

public:

Rhombus();

Rhombus(double a, double b);

Rhombus(std::istream& is);

double Area() override;

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

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

virtual ~Rhombus();

private:

double side;

double height;

};

#endif

**Trapezium.h**

#ifndef TRAPEZIUM\_H

#define TRAPEZIUM\_H

#include <iostream>

#include "Figure.h"

class Trapezium : public Figure{

public:

Trapezium();

Trapezium(double a, double b, double c);

Trapezium(std::istream& is);

double Area() override;

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

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

virtual ~Trapezium();

private:

double side\_a;

double side\_b;

double height;

};

#endif

**Cont.h**

#ifndef CONT.H

#define CONT.H

#include <memory>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "ContItem.h"

#include "Iterator.cpp"

const int size = 10;

template <class T1, class T2, class T3>

class Cont{

public:

Cont(int size);

void push(std::shared\_ptr<T1> && t, int index);

void push(std::shared\_ptr<T2> && t, int index);

void push(std::shared\_ptr<T3> && t, int index);

bool IsT1(int index);

bool IsT2(int index);

bool IsT3(int index);

std::shared\_ptr<T1> GetT1(int index);

std::shared\_ptr<T2> GetT2(int index);

std::shared\_ptr<T3> GetT3(int index);

void removeIt(int index);

void removeIt\_area(double area\_);

Iterator<T1, T2, T3> begin();

Iterator<T1, T2, T3> end();

template <class A1, class A2, class A3> friend std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont);

virtual ~Cont();

private:

ContItem<T1, T2, T3> \*a;

int size\_;

Iterator<T1, T2, T3> first;

Iterator<T1, T2, T3> last;

};

#endif

**ContItem.h**

#ifndef CONTITEM\_H

#define CONTITEM\_H

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include "Allocator.h"

#include <memory>

#include <iostream>

template <class T1, class T2, class T3>

class ContItem{

public:

ContItem();

ContItem(int index);

ContItem(std::shared\_ptr<T1> &t, int index);

ContItem(std::shared\_ptr<T2> &t, int index);

ContItem(std::shared\_ptr<T3> &t, int index);

bool IsT1() const;

bool IsT2() const;

bool IsT3() const;

std::shared\_ptr<T1> GetT1();

std::shared\_ptr<T2> GetT2();

std::shared\_ptr<T3> GetT3();

void SetIndex(int index);

int GetIndex();

double Area\_();

void \*operator new(size\_t size);

void operator delete(void \*ptr);

template <class A1, class A2, class A3> friend std::ostream& operator<< (std::ostream & os, const ContItem<A1, A2, A3> & item);

virtual ~ContItem();

private:

std::shared\_ptr<T1> t1;

std::shared\_ptr<T2> t2;

std::shared\_ptr<T3> t3;

int index\_;

static Allocator arritem\_allocator;

};

#endif

**Iterator.h**

#ifndef ITERATOR\_H

#define ITERATOR\_H

#include <memory>

#include <iostream>

#include "ContItem.cpp"

template<class T1, class T2, class T3>

class Iterator

{

public:

Iterator();

Iterator(ContItem<T1, T2, T3>\* p);

ContItem<T1, T2, T3>\* operator \*();

bool operator==(Iterator i);

bool operator != (Iterator i);

void operator ++ ();

private:

ContItem<T1, T2, T3> \* el;

};

#endif

**Allocator.h**

#ifndef ALLOCATOR\_H

#define ALLOCATOR\_H

#include <cstdlib>

class Allocator {

public:

Allocator(size\_t size,size\_t count);

void \*allocate();

void deallocate(void \*pointer);

bool has\_free\_blocks();

virtual ~Allocator();

private:

size\_t \_size;

size\_t \_count;

char \*\_used\_blocks;

void \*\*\_free\_blocks;

size\_t \_free\_count;

};

#endif

**BTree.h**

#ifndef BTREE\_H

#define BTREE\_H

#include <memory>

#include <future>

#include <mutex>

#include <thread>

#include <iostream>

#include "BTreeItem.h"

template <class T>

class BTree{

public:

BTree();

BTreeItem<T> \* push(BTreeItem<T>\* root, T value);

BTreeItem<T> \* root;

size\_t size\_;

private:

};

#endif // BTREE\_H

**BTreeItem.h**

#ifndef BTREEITEM\_H

#define BTREEITEM\_H

#include <thread>

const int max\_ = 2;

template <class T>

class BTreeItem

{

public:

BTreeItem();

BTreeItem(T t);

T value;

BTreeItem<T>\* children[max\_];

private:

};

#endif

**Rectangle.cpp**

#include "Rectangle.h"

#include <iostream>

Rectangle::Rectangle(){

side\_a=0.0;

side\_b=0.0;

}

Rectangle::Rectangle(double a,double b){

side\_a=a;

side\_b=b;

}

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

is>>\*this;

}

double Rectangle::Area(){

return side\_a\*side\_b;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0) obj.err=true;

return is;

}

Rectangle::~Rectangle(){

//std::cout << "Прямоугольник удален" << std::endl;

}

**Rhombus.cpp**

#include "Rhombus.h"

#include <iostream>

Rhombus::Rhombus(){

side=0.0;

height=0.0;

}

Rhombus::Rhombus(double a,double b){

side=a;

height=b;

}

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

is>>\*this;

}

double Rhombus::Area(){

return side\*height;

}

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

os<<"Сторона = "<<obj.side<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону: "<<std::endl;

if(!(is>>obj.side)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side<0 || obj.height<0) obj.err=true;

return is;

}

Rhombus::~Rhombus(){

//std::cout << "Ромб удален" << std::endl;

}

**Trapezium.cpp**

#include "Trapezium.h"

#include <iostream>

Trapezium::Trapezium(){

side\_a=0.0;

side\_b=0.0;

height=0.0;

}

Trapezium::Trapezium(double a, double b, double c){

side\_a=a;

side\_b=b;

height=c;

}

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

is>>\*this;

}

double Trapezium::Area(){

return (side\_a+side\_b)/2.0\*height;

}

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

os<<"Сторона а= "<<obj.side\_a<<", Сторона b= "<<obj.side\_b<<", Высота = "<<obj.height;

return os;

}

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

std::cout<<"Введите сторону a: "<<std::endl;

if(!(is>>obj.side\_a)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите сторону b: "<<std::endl;

if(!(is>>obj.side\_b)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

std::cout<<"Введите высоту: "<<std::endl;

if(!(is>>obj.height)){

is.clear();

while(is.get() !='\n');

obj.err=true;

return is;

}

if(obj.side\_a<0 || obj.side\_b<0 || obj.height<0) obj.err=true;

return is;

}

Trapezium::~Trapezium(){

//std::cout << "Трапеция удален" << std::endl;

}

**Cont.cpp**

#include "Cont.h"

#include <iostream>

#include <algorithm>

template <class T1, class T2, class T3> Cont<T1, T2, T3>::Cont(int size){

a = new ContItem<T1, T2, T3>[size];

size\_ = size;

for (int i = 0; i < size; i++) a[i].SetIndex(i);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T1> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T2> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::push(std::shared\_ptr<T3> && t, int index){

a[index]=ContItem<T1, T2, T3>(t, index);

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT1(int index){

return a[index].IsT1();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT2(int index){

return a[index].IsT2();

}

template <class T1, class T2, class T3> bool Cont<T1, T2, T3>::IsT3(int index){

return a[index].IsT3();

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> Cont<T1, T2, T3>::GetT1(int index){

return a[index].GetT1();

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> Cont<T1, T2, T3>::GetT2(int index){

return a[index].GetT2();

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> Cont<T1, T2, T3>::GetT3(int index){

return a[index].GetT3();

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::removeIt(int index){

a[index] = ContItem<T1, T2, T3>();

}

template <class T1, class T2, class T3> void Cont<T1, T2, T3>::removeIt\_area(double area\_){

for (int i = 0; i < size\_; i++){

if (a[i].Area\_() < area\_) removeIt(i);

}

}

template <class A1, class A2, class A3> std::ostream& operator <<(std::ostream & os,const Cont<A1, A2, A3> & cont){

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

os << "[" << i << "] - " << cont.a[i] << std::endl;

}

return os;

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::begin()

{

return Iterator<T1, T2, T3>(a);

}

template <class T1, class T2, class T3> Iterator<T1, T2, T3> Cont<T1, T2, T3>::end()

{

return Iterator<T1, T2, T3>(&a[size\_]);

}

template <class T1, class T2, class T3> Cont<T1, T2, T3>::~Cont(){

for (int i = 0; i < size; i++) a[i] = ContItem<T1, T2, T3>();

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class Cont<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const Cont<Rectangle, Rhombus, Trapezium> & cont);

**ContItem.cpp**

#include "ContItem.h"

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem() : t1(nullptr), t2(nullptr), t3(nullptr) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(int index) : t1(nullptr), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T1> &t, int index) : t1(t), t2(nullptr), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T2> &t, int index) : t1(nullptr), t2(t), t3(nullptr), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::ContItem(std::shared\_ptr<T3> &t, int index) : t1(nullptr), t2(nullptr), t3(t), index\_(index) {}

template <class T1, class T2, class T3> ContItem<T1, T2, T3>::~ContItem() {}

template <class T1, class T2, class T3> Allocator ContItem<T1, T2, T3>::arritem\_allocator(sizeof(ContItem<T1, T2, T3>), 10);

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT1() const

{

if (t1 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT2() const

{

if (t2 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> bool ContItem<T1, T2, T3>::IsT3() const

{

if (t3 != nullptr) return true;

else return false;

}

template <class T1, class T2, class T3> std::shared\_ptr<T1> ContItem<T1, T2, T3>::GetT1()

{

return this->t1;

}

template <class T1, class T2, class T3> std::shared\_ptr<T2> ContItem<T1, T2, T3>::GetT2()

{

return this->t2;

}

template <class T1, class T2, class T3> std::shared\_ptr<T3> ContItem<T1, T2, T3>::GetT3()

{

return this->t3;

}

template <class T1, class T2, class T3> void ContItem<T1, T2, T3>::SetIndex(int index)

{

index\_ = index;

}

template <class T1, class T2, class T3> int ContItem<T1, T2, T3>::GetIndex()

{

return index\_;

}

template <class T1, class T2, class T3> double ContItem<T1, T2, T3>::Area\_()

{

if(t1) return t1->Area();

else if(t2) return t2->Area();

else if(t3) return t3->Area();

else return 0;

}

template <class T1, class T2, class T3> void \* ContItem<T1, T2, T3>::operator new(size\_t size){

return arritem\_allocator.allocate();

}

template <class T1, class T2, class T3> void ContItem<T1, T2, T3>::operator delete(void \*ptr){

arritem\_allocator.deallocate(ptr);

}

template <class A1, class A2, class A3> std::ostream& operator << (std::ostream &os, const ContItem<A1, A2, A3> &item)

{

if (item.IsT1())

os << \*item.t1 << " (прямоугольник)";

else if (item.IsT2())

os << \*item.t2 << " (ромб)";

else if (item.IsT3())

os << \*item.t3 << " (трапеция)";

else

os << "Пусто";

return os;

}

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

template class ContItem<Rectangle, Rhombus, Trapezium>;

template std::ostream& operator <<(std::ostream & os,const ContItem<Rectangle, Rhombus, Trapezium> & item);

**Iterator.cpp**

#include "Iterator.h"

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator()

{

el = nullptr;

}

template<class T1, class T2, class T3>

Iterator<T1, T2, T3>::Iterator(ContItem<T1, T2, T3>\* p)

{

el = p;

}

template<class T1, class T2, class T3>

ContItem<T1, T2, T3>\* Iterator<T1, T2, T3>::operator \* ()

{

return el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator ==(Iterator i)

{

return el == i.el;

}

template<class T1, class T2, class T3>

bool Iterator<T1, T2, T3>::operator !=(Iterator i)

{

return !(\*this == i);

}

template<class T1, class T2, class T3>

void Iterator<T1, T2, T3>::operator ++ ()

{

el++;

}

**Allocator.cpp**

#include "Allocator.h"

#include <iostream>

Allocator::Allocator(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\_count = \_count;

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

}

void \*Allocator::allocate() {

void \*result = nullptr;

if(\_free\_count>0){

result = \_free\_blocks[\_free\_count-1];

\_free\_count--;

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

}

else{

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

}

return result;

}

void Allocator::deallocate(void \*pointer) {

std::cout << "Allocator: Deallocate"<< std::endl;

\_free\_blocks[\_free\_count] = pointer;

\_free\_count ++;

}

bool Allocator::has\_free\_blocks() {

return \_free\_count>0;

}

Allocator::~Allocator() {

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

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

delete \_free\_blocks;

delete \_used\_blocks;

}

**BTree.cpp**

#include "BTree.h"

#include <queue>

#include <iostream>

template <typename T> BTree<T>::BTree(){

root = nullptr;

size\_ = 0;

}

template <typename T> BTreeItem<T> \* BTree<T>::push(BTreeItem<T> \* root, T value){

std::queue <BTreeItem<T>\*> nodes\_to\_visit;

BTreeItem<T>\* current\_node;

if (!root)

{

root = new BTreeItem<T>(value);

return root;

}

else

{

nodes\_to\_visit.push(root);

}

while (!nodes\_to\_visit.empty())

{

current\_node = nodes\_to\_visit.front();

nodes\_to\_visit.pop();

for (int i = 0; i<max\_; i++)

{

if (!current\_node->children[i])

{

current\_node->children[i] = new BTreeItem<T>(value);

return root;

}

else nodes\_to\_visit.push(current\_node->children[i]);

}

}

/\*BTreeItem<T> \* val = new BTreeItem<T>(value);

if (!root){

root = val;

size\_++;

return root;

}

else if(!root->leftT){

root->leftT = val;

return root;

}

else if(!root->rightT){

root->rightT = val;

return root;

}

else if(size\_<6){

val = push(root->leftT, value);

root->leftT = val;

}

else if(size\_>5){

val = push(root->rightT, value);

root->rightT = val;

} \*/

}

#include <functional>

template class BTree<std::function<void(void)>>;

**BTreeItem.cpp**

#include "BTreeItem.h"

template <typename T> BTreeItem<T>::BTreeItem() : value(nullptr) {

for (int i = 0; i < max\_; i++) children[i] = nullptr;

}

template <typename T> BTreeItem<T>::BTreeItem(T t) : value(t) {

for (int i = 0; i < max\_; i++) children[i] = nullptr;

}

#include <functional>

template class BTreeItem<std::function<void(void)>>;

**Main.cpp**

#include <iostream>

#include <clocale>

#include "Rectangle.h"

#include "Trapezium.h"

#include "Rhombus.h"

#include "Figure.h"

#include <memory>

#include <queue>

#include "ContItem.h"

#include "Cont.h"

#include "Allocator.h"

#include "BTree.h"

#include "BTreeItem.h"

#include <cstdlib>

#include <random>

#include <functional>

#include <future>

#include <thread>

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

Cont<Rectangle, Rhombus, Trapezium> container(10);

BTree<command> tree;

void Circumvention(BTreeItem<command> \*root){

std::queue <BTreeItem<command>\*> nodes\_to\_visit;

BTreeItem<command>\* current\_node;

nodes\_to\_visit.push(root);

while (!nodes\_to\_visit.empty())

{

current\_node = nodes\_to\_visit.front();

nodes\_to\_visit.pop();

command cmd = current\_node->value;

cmd();

for (int i = 0; i < max\_; i++)

{

if (current\_node->children[i])

{

nodes\_to\_visit.push(current\_node->children[i]);

}

}

}

}

int main()

{

setlocale(LC\_CTYPE,"rus");

int pos = 0;

command cmd\_push = [&]()

{

int n;

std::cout << "Колличество фигур: ";

std::cin >> n;

for (int i = 0; i < n; i++)

{

int figure\_number = rand() % 3;

int a = rand() % 10;

int b = rand() % 10;

int c = rand() % 10;

if (figure\_number == 0)

container.push(std::shared\_ptr<Rectangle>(new Rectangle(a, b)), pos);

else if (figure\_number == 1)

container.push(std::shared\_ptr<Rhombus>(new Rhombus(a, b)), pos);

else if (figure\_number == 2)

container.push(std::shared\_ptr<Trapezium>(new Trapezium(a, b, c)), pos);

pos++;

}

};

command cmd\_remove\_by\_area = [&]()

{

double area;

std::cout << "Введите площадь: ";

std::cin >> area;

container.removeIt\_area(area);

};

command cmd\_print = [&]()

{

std::cout << "Массив:\n" << container << std::endl;

};

tree.root = tree.push(tree.root, cmd\_push);

tree.root = tree.push(tree.root, cmd\_push);

tree.root = tree.push(tree.root, cmd\_push);

tree.root = tree.push(tree.root, cmd\_print);

tree.root = tree.push(tree.root, cmd\_remove\_by\_area);

tree.root = tree.push(tree.root, cmd\_print);

Circumvention(tree.root);

return 0;

}

**Вывод:**

Allocator: Memory init

Колличество фигур: 3

Колличество фигур: 2

Колличество фигур: 2

Массив:

[0] - Сторона а= 7, Сторона b= 4, Высота = 0 (трапеция)

[1] - Сторона а= 4, Сторона b= 8, Высота = 8 (трапеция)

[2] - Сторона = 4, Высота = 5 (ромб)

[3] - Сторона = 7, Высота = 1 (ромб)

[4] - Сторона = 2, Высота = 7 (ромб)

[5] - Сторона а= 4, Сторона b= 2 (прямоугольник)

[6] - Сторона = 2, Высота = 1 (ромб)

[7] - Пусто

[8] - Пусто

[9] - Пусто

Введите площадь: 2

Массив:

[0] - Пусто

[1] - Сторона а= 4, Сторона b= 8, Высота = 8 (трапеция)

[2] - Сторона = 4, Высота = 5 (ромб)

[3] - Сторона = 7, Высота = 1 (ромб)

[4] - Сторона = 2, Высота = 7 (ромб)

[5] - Сторона а= 4, Сторона b= 2 (прямоугольник)

[6] - Сторона = 2, Высота = 1 (ромб)

[7] - Пусто

[8] - Пусто

[9] - Пусто

Allocator: Memory freed

Process returned 0 (0x0) execution time : 13.260 s

Press any key to continue.