Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Объектно-ориентированное программирование»

III семестр

Задание 6: «Основные работы с коллекциями: итераторы»

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №12 |
| Студент: | Коростелев Дмитрий Васильевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 09.12.2019 |

Москва, 2019

1. **Задание**

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться oop\_exercise\_06 (в случае использования Windows oop\_exercise\_06.exe)

Необходимо зарегистрироваться на GitHub (если студент уже имеет регистрацию на GitHub то можно использовать ее) и создать репозитарий для задания лабораторной работы.

Преподавателю необходимо предъявить ссылку на публичный репозиторий на Github. Имя репозитория должно быть <https://github.com/login/oop_exercise_06>

Создать шаблон динамической коллекцию, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей (std::shared\_ptr, std::weak\_ptr).

Опционально использование std::unique\_ptr;

2. В качестве параметра шаблона коллекция должна принимать тип данных;

3. Коллекция должна содержать метод доступа:

Стек – pop, push, top;

Очередь – pop, push, top;

Список,Динамический массив – доступ к элементу по оператору [];

Реализовать аллокатор, который выделяет фиксированный размер памяти (количество блоков памяти – является параметром шаблона аллокатора). Внутри аллокатор должен хранить указатель на используемый блок памяти и динамическую коллекцию указателей на свободные блоки. Динамическая коллекция должна соответствовать варианту задания (Динамический массив, Список, Стек, Очередь);

Коллекция должна использовать аллокатор для выделеления и освобождения памяти для своих элементов.

Аллокатор должен быть совместим с контейнерами std::map и std::list (опционально – vector).

Реализовать программу, которая: Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию использующую аллокатор; Позволяет удалять элемент из коллекции по номеру элемента; Выводит на экран введенные фигуры c помощью std::for\_each;

1. **Адрес репозитория на GitHub**

[https://github.com/Dmitry4K/oop\_exercise\_](https://github.com/Dmitry4K/oop_exercise_2)06

1. **Код программы на С++**

*main.cpp*

#include<iostream>

#include<algorithm>

#include<locale.h>

#include"trapeze.h"

#include"containers/list.h"

#include"allocators/allocator.h"

void Menu1() {

std::cout << "1. Добавить фигуру в список\n";

std::cout << "2. Удалить фигуру\n";

std::cout << "3. Вывести фигуру\n";

std::cout << "4. Вывести все фигуры\n";

std::cout << "5. Вывести кол-во фигур чья площаль больше чем ...\n";

}

void PushMenu() {

std::cout << "1. Добавить фигуру в начало списка\n";

std::cout << "2. Добавить фигуру в конец списка\n";

std::cout << "3. Добавить фигуру по индексу\n";

}

void DeleteMenu() {

std::cout << "1. Удалить фигуру в начале списка\n";

std::cout << "2. Удалить фигуру в конце списка\n";

std::cout << "3. Удалить фигуру по индексу\n";

}

void PrintMenu() {

std::cout << "1. Вывести первую фигуру в списке\n";

std::cout << "2. Вывести последнюю фигуру в списке\n";

std::cout << "3. Вывести фигуру по индексу\n";

}

int main() {

setlocale(LC\_ALL, "rus");

containers::list<Trapeze<int>, allocators::my\_allocator<Trapeze<int>, 500>> MyList;

/\*

MyList.push\_back(1);

MyList.push\_back(2);

MyList.push\_back(3);

MyList.push\_front(4);

MyList.pop\_front();

MyList.pop\_back();

for (auto El : MyList)

std::cout << El<< ' ';

std::cout << std::endl;

std::cout << MyList.front() << std::endl;

std::cout << MyList[1] << std::endl;

std::cout << MyList.back() << std::endl;

\*/

Trapeze<int> TempTrapeze;

/\*

TempTrapeze.read(std::cin);

MyList.push\_front(TempTrapeze);

std::cout << MyList.length() << std::endl;

\*/

//\*

while (true) {

Menu1();

int n, m, ind;

double s;

std::cin >> n;

switch (n) {

case 1:

TempTrapeze.read(std::cin);

PushMenu();

std::cin >> m;

switch (m) {

case 1:

MyList.push\_front(TempTrapeze);

break;

case 2:

MyList.push\_back(TempTrapeze);

break;

case 3:

std::cin >> ind;

MyList.insert\_by\_number(ind, TempTrapeze);

default:

break;

}

break;

case 2:

DeleteMenu();

std::cin >> m;

switch (m) {

case 1:

MyList.pop\_front();

break;

case 2:

MyList.pop\_back();

break;

case 3:

std::cin >> ind;

MyList.delete\_by\_number(ind);

break;

default:

break;

}

break;

case 3:

PrintMenu();

std::cin >> m;

switch (m) {

case 1:

MyList.front().print(std::cout);

std::cout << std::endl;

break;

case 2:

MyList.back().print(std::cout);

std::cout << std::endl;

break;

case 3:

std::cin >> ind;

MyList[ind].print(std::cout);

std::cout << std::endl;

break;

default:

break;

}

break;

case 4:

std::for\_each(MyList.begin(), MyList.end(), [](Trapeze<int> &X) { X.print(std::cout); std::cout << std::endl; });

/\*for (auto Element : MyList) {

Element.print(std::cout);

std::cout << std::endl;

}\*/

break;

case 5:

std::cin >> s;

std::cout << std::count\_if(MyList.begin(), MyList.end(), [=](Trapeze<int>& X) {return X.square() > s; }) << std::endl;

break;

default:

return 0;

}

}

//\*/

system("pause");

return 0;

}

*vertex.h*

#pragma once

#include<iostream>

#include<cmath>

template<class T>

class Vertex {

public:

T x, y;

//Vertex<T>& Vertex<T>::operator=(const Vertex<T>& a);

};

template<class T>

std::istream& operator>>(std::istream& is, Vertex<T>& point) {

is >> point.x >> point.y;

return is;

}

template<class T>

std::ostream& operator<<(std::ostream& os, Vertex<T> point) {

os <<'['<< point.x << ", " << point.y<<']';

return os;

}

template<class T>

Vertex<T> operator+(const Vertex<T>& a, const Vertex<T>& b) {

Vertex<T> res;

res.x = a.x + b.x;

res.y = a.y + b.y;

return res;

}

/\*

template<class T>

Vertex<T>& Vertex<T>::operator=(const Vertex<T>& a) {

x = a.x;

y = a.y;

return \*this;

}\*/

template<class T>

Vertex<T> operator+=(Vertex<T> &a, const Vertex<T> &b) {

a.x += b.x;

a.y += b.y;

return a;

}

template<class T>

double distance(const Vertex<T> &a, const Vertex<T>& b) {

return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));

}

*trapeze.h*

#pragma once

#include"vertex.h"

template <class T>

class Trapeze {

private:

Vertex<T> Vertexs[4];

public:

using vertex\_type = Vertex<T>;

Trapeze();

Trapeze(std::istream& in);

void read(std::istream& in);

Vertex<T> center() const;

double square() const;

void print(std::ostream& os) const;

friend std::ostream& operator<< (std::ostream &out, const Trapeze<T> &point);

friend std::ostream& operator>> (std::istream &in, const Trapeze<T> &point);

};

template<class T> Trapeze<T>::Trapeze() {}

template<class T> Trapeze<T>::Trapeze(std::istream& in) {

for (int i = 0; i < 4; i++)

in >> Vertexs[i];

}

template<class T> double Trapeze<T>::square() const {

double Area = 0;

for (int i = 0; i < 4; i++) {

Area += (Vertexs[i].x) \* (Vertexs[(i + 1) % 4].y) - (Vertexs[(i + 1) % 4].x)\*(Vertexs[i].y);

}

Area \*= 0.5;

return abs(Area);

}

template<class T> void Trapeze<T>::print(std::ostream& os) const {

os << "Trapeze: ";

for (int i = 0; i < 4; i++)

os << Vertexs[i] << ' ';

os << '\n';

}

template<class T> Vertex<T> Trapeze<T>::center() const {

Vertex<T> res = Vertex<T>();

for (int i = 0; i < 4; i++)

res += Vertexs[i];

return res / 4;

}

template <class T> void Trapeze<T>::read(std::istream& in) {

Trapeze<T> res = Trapeze(in);

\*this = res;

}

template<class T>

std::ostream& operator<< (std::ostream &out, const Trapeze<T> &point) {

out << "Trapeze: ";

for (int i = 0; i < 4; i++)

out << point.Vertexs[i] << ' ';

out << '\n';

}

template<class T>

std::ostream& operator>> (std::istream &in, const Trapeze<T> &point){

for (int i = 0; i < 4; i++)

in >> point.Vertexs[i];

}

*list.h*

#pragma once

#include <iterator>

#include <memory>

namespace containers {

template<class T, class Allocator = std::allocator<T>>

class list {

private:

struct element;//объявление типа хранящегося в list, для того, чтобы он был виден forward\_iterator

size\_t size = 0;//размер списка

public:

list() = default;//коструктор по умолчанию

class forward\_iterator {

public:

using value\_type = T;

using reference = value\_type& ;

using pointer = value\_type\* ;

using difference\_type = std::ptrdiff\_t;

using iterator\_category = std::forward\_iterator\_tag;

explicit forward\_iterator(element\* ptr);

T& operator\*();

forward\_iterator& operator++();

forward\_iterator operator++(int);

bool operator== (const forward\_iterator& other) const;

bool operator!= (const forward\_iterator& other) const;

private:

element\* it\_ptr;

friend list;

};

forward\_iterator begin();

forward\_iterator end();

void push\_back(const T& value);

void push\_front(const T& value);

T& front();

T& back();

void pop\_back();

void pop\_front();

size\_t length();

bool empty();

void delete\_by\_it(forward\_iterator d\_it);

void delete\_by\_number(size\_t N);

void insert\_by\_it(forward\_iterator ins\_it, T& value);

void insert\_by\_number(size\_t N, T& value);

list& operator=(list& other);

T& operator[](size\_t index);

private:

using allocator\_type = typename Allocator::template rebind<element>::other;

struct deleter {

private:

allocator\_type\* allocator\_;

public:

deleter(allocator\_type\* allocator) : allocator\_(allocator) {}

void operator() (element\* ptr) {

if (ptr != nullptr) {

std::allocator\_traits<allocator\_type>::destroy(\*allocator\_, ptr);

allocator\_->deallocate(ptr, 1);

}

}

};

using unique\_ptr = std::unique\_ptr<element, deleter>;

struct element {

T value;

unique\_ptr next\_element = { nullptr, deleter{nullptr} };

element\* prev\_element = nullptr;

element(const T& value\_) : value(value\_) {}

forward\_iterator next();

};

allocator\_type allocator\_{};

unique\_ptr first{ nullptr, deleter{nullptr} };

element\* tail = nullptr;

};

template<class T, class Allocator>

typename list<T, Allocator>::forward\_iterator list<T, Allocator>::begin() {//+

return forward\_iterator(first.get());

}

template<class T, class Allocator>

typename list<T, Allocator>::forward\_iterator list<T, Allocator>::end() {//+

return forward\_iterator(nullptr);

}

template<class T, class Allocator>

size\_t list<T, Allocator>::length() {//+

return size;

}

template<class T, class Allocator>

bool list<T, Allocator>::empty() {

return length() == 0;

}

template<class T, class Allocator>

void list<T, Allocator>::push\_back(const T& value) {

element\* result = this->allocator\_.allocate(1);

std::allocator\_traits<allocator\_type>::construct(this->allocator\_, result, value);

if (!size) {

first = unique\_ptr(result, deleter{ &this->allocator\_ });

tail = first.get();

size++;

return;

}

tail->next\_element = unique\_ptr(result, deleter{ &this->allocator\_ });

element\* temp = tail;//?

tail = tail->next\_element.get();

tail->prev\_element = temp;//?

size++;

//first = push\_impl(std::move(first),nullptr, value);

//size++;

}

template<class T, class Allocator>

void list<T, Allocator>::push\_front(const T& value) {

size++;

element\* result = this->allocator\_.allocate(1);

std::allocator\_traits<allocator\_type>::construct(this->allocator\_, result, value);

unique\_ptr tmp = std::move(first);

first = unique\_ptr(result, deleter{ &this->allocator\_ });

first->next\_element = std::move(tmp);

if(first->next\_element)

first->next\_element->prev\_element = first.get();

if (size == 1) {

tail = first.get();

}

if (size == 2) {

tail = first->next\_element.get();

}

}

template<class T, class Allocator>

void list<T, Allocator>::pop\_front() {

if (size == 0) {

throw std::logic\_error("can`t pop from empty list");

}

if (size == 1) {

first = nullptr;

tail = nullptr;

size--;

return;

}

first = std::move(first->next\_element);

if (first->prev\_element)

first->prev\_element = nullptr;

size--;

if (size == 1)

tail = first.get();

}

template<class T, class Allocator>

void list<T, Allocator>::pop\_back() {

if (size == 0) {

throw std::logic\_error("can`t pop from empty list");

}

element\* tmp = nullptr;

if (tail->prev\_element){

tmp = tail->prev\_element;

//unique\_ptr dump = std::move(tail->prev\_element->next\_element);

tail->prev\_element->next\_element = nullptr;

}

else{

tmp = first.get();

unique\_ptr dump = std::move(first);

}

tail = tmp;

//first = pop\_impl(std::move(first));

size--;

}

template<class T, class Allocator>

T& list<T, Allocator>::front() {

if (size == 0) {

throw std::logic\_error("list is empty");

}

return first->value;

}

template<class T, class Allocator>

T& list<T, Allocator>::back() {

if (size == 0) {

throw std::logic\_error("list is empty");

}

forward\_iterator i = this->begin();

while ( i.it\_ptr->next() != this->end()) {

i++;

}

return \*i;

}

template<class T, class Allocator>

list<T,Allocator>& list<T, Allocator>::operator=(list<T, Allocator>& other) {

size = other.size;

first = std::move(other.first);

}

template<class T, class Allocator>

void list<T, Allocator>::delete\_by\_it(containers::list<T, Allocator>::forward\_iterator d\_it) {

forward\_iterator i = this->begin(), end = this->end();

if (d\_it == end) throw std::logic\_error("out of borders");

if (d\_it == this->begin()) {

this->pop\_front();

return;

}

if (d\_it.it\_ptr == tail) {

this->pop\_back();

return;

}

/\*

while ((i.it\_ptr != nullptr) && (i.it\_ptr->next() != d\_it)) {

++i;

}

\*/

if (d\_it.it\_ptr == nullptr) throw std::logic\_error("out of broders");

auto temp = d\_it.it\_ptr->prev\_element;

d\_it.it\_ptr->prev\_element->next\_element = std::move(d\_it.it\_ptr->next\_element);

d\_it.it\_ptr = d\_it.it\_ptr->prev\_element;

d\_it.it\_ptr->next\_element->prev\_element = temp;

/\*

if (i.it\_ptr == nullptr) throw std::logic\_error("out of borders");

i.it\_ptr->next\_element = std::move(d\_it.it\_ptr->next\_element);

\*/

size--;

}

template<class T, class Allocator>

void list<T, Allocator>::delete\_by\_number(size\_t N) {

forward\_iterator it = this->begin();

for (size\_t i = 0; i < N; ++i) {

++it;

}

this->delete\_by\_it(it);

}

template<class T, class Allocator>

void list<T, Allocator>::insert\_by\_it(containers::list<T, Allocator>::forward\_iterator ins\_it, T& value) {

//auto tmp = unique\_ptr(new element{ value });

element\* tmp = this->allocator\_.allocate(1);

std::allocator\_traits<allocator\_type>::construct(this->allocator\_, tmp, value);

forward\_iterator i = this->begin();

if (ins\_it == this->begin()) {

this->push\_front(value);

/\* tmp->next\_element = std::move(first);

first = unique\_ptr(tmp, deleter{ &this->allocator\_ });

if(first->next\_element)

first->next\_element->prev\_element = first.get();

//??????

\*/

//size++;

return;

}

if(ins\_it.it\_ptr == tail){

this->push\_back(value);

}

/\*

while ((i.it\_ptr != nullptr) && (i.it\_ptr->next() != ins\_it)) {

++i;

}

/\*

if(ins\_it.it\_ptr->prev\_element == nullptr) throw std::logic\_error("out of borders");

auto tmp1 = ins\_it.it\_ptr->prev\_element;

tmp->next\_element = std::move(ins\_it.it\_ptr->prev\_element->next\_element);

tmp->prev\_element = tmp1;

tmp->next\_element->prev\_element = tmp.get();

ins\_it.it\_ptr->prev\_element->next\_element = std::move(tmp);

\*/

tmp->prev\_element = ins\_it.it\_ptr->prev\_element;

ins\_it.it\_ptr->prev\_element = tmp;

//unique\_ptr(result, deleter{ &this->allocator\_ });

tmp->next\_element = unique\_ptr(ins\_it.it\_ptr, deleter{ &this->allocator\_ });

tmp->prev\_element->next\_element = unique\_ptr(tmp, deleter{ &this->allocator\_ });

size++;

}

template<class T, class Allocator>

void list<T, Allocator>::insert\_by\_number(size\_t N, T& value) {

forward\_iterator it = this->begin();

for (size\_t i = 0; i < N; ++i) {

++it;

}

this->insert\_by\_it(it, value);

}

template<class T, class Allocator>

typename list<T,Allocator>::forward\_iterator list<T, Allocator>::element::next() {

return forward\_iterator(this->next\_element.get());

}

template<class T, class Allocator>

list<T, Allocator>::forward\_iterator::forward\_iterator(containers::list<T, Allocator>::element \*ptr) {

it\_ptr = ptr;

}

template<class T, class Allocator>

T& list<T, Allocator>::forward\_iterator::operator\*() {

return this->it\_ptr->value;

}

template<class T, class Allocator>

T& list<T, Allocator>::operator[](size\_t index) {

if (index < 0 || index >= size) {

throw std::out\_of\_range("out of list's borders");

}

forward\_iterator it = this->begin();

for (size\_t i = 0; i < index; i++) {

it++;

}

return \*it;

}

template<class T, class Allocator>

typename list<T, Allocator>::forward\_iterator& list<T, Allocator>::forward\_iterator::operator++() {

if (it\_ptr == nullptr) throw std::logic\_error("out of list borders");

\*this = it\_ptr->next();

return \*this;

}

template<class T, class Allocator>

typename list<T, Allocator>::forward\_iterator list<T, Allocator>::forward\_iterator::operator++(int) {

forward\_iterator old = \*this;

++\*this;

return old;

}

template<class T, class Allocator>

bool list<T, Allocator>::forward\_iterator::operator==(const forward\_iterator& other) const {

return it\_ptr == other.it\_ptr;

}

template<class T, class Allocator>

bool list<T, Allocator>::forward\_iterator::operator!=(const forward\_iterator& other) const {

return it\_ptr != other.it\_ptr;

}

}

*allocator.h*

|  |
| --- |
| #pragma once |
|  |  |
|  | #include <cstdlib> |
|  | #include <iostream> |
|  | #include <type\_traits> |
|  | #include <queue> |
|  |  |
|  | namespace allocators { |
|  |  |
|  | template<class T, size\_t ALLOC\_SIZE>//ALLOC\_SIZE - размер, который требуется выделить |
|  | struct my\_allocator { |
|  |  |
|  | private: |
|  | char\* pool\_begin; //указатель на начало хранилища |
|  | char\* pool\_end;//указатель на конец хранилища pool\_begin pool\_tail pool\_end |
|  | char\* pool\_tail;//указатель на конец заполненного пространства |1000111010100101|----------------------------------------| |
|  | std::queue<char\*> free\_blocks; |
|  | public: |
|  | using value\_type = T; |
|  | using size\_type = std::size\_t; |
|  | using difference\_type = std::ptrdiff\_t; |
|  | using is\_always\_equal = std::false\_type; |
|  |  |
|  | template<class U> |
|  | struct rebind { |
|  | using other = my\_allocator<U, ALLOC\_SIZE>; |
|  | }; |
|  |  |
|  | my\_allocator() : |
|  | pool\_begin(new char[ALLOC\_SIZE]), |
|  | pool\_end(pool\_begin + ALLOC\_SIZE), |
|  | pool\_tail(pool\_begin) |
|  | {} |
|  |  |
|  | my\_allocator(const my\_allocator&) = delete; |
|  | my\_allocator(my\_allocator&&) = delete; |
|  |  |
|  | ~my\_allocator() { |
|  | delete[] pool\_begin; |
|  | } |
|  |  |
|  | T\* allocate(std::size\_t n); |
|  | void deallocate(T\* ptr, std::size\_t n); |
|  |  |
|  | }; |
|  |  |
|  | template<class T, size\_t ALLOC\_SIZE> |
|  | T\* my\_allocator<T, ALLOC\_SIZE>::allocate(std::size\_t n) { |
|  | if (n != 1) { |
|  | throw std::logic\_error("can`t allocate arrays"); |
|  | } |
|  | if (size\_t(pool\_end - pool\_tail) < sizeof(T)) { |
|  | if (free\_blocks.size()) {//ищем свободное место в райне отданном пространстве |
|  | //auto it = free\_blocks.; |
|  | char\* ptr = free\_blocks.front(); |
|  | free\_blocks.pop(); |
|  | return reinterpret\_cast<T\*>(ptr); |
|  | } |
|  | throw std::bad\_alloc();//èíà÷å bad\_alloc |
|  | } |
|  | T\* result = reinterpret\_cast<T\*>(pool\_tail);//приведение к типу |
|  | pool\_tail += sizeof(T); |
|  | return result; |
|  | } |
|  |  |
|  | template<class T, size\_t ALLOC\_SIZE> |
|  | void my\_allocator<T, ALLOC\_SIZE>::deallocate(T\* ptr, std::size\_t n) { |
|  | if (n != 1) { |
|  | throw std::logic\_error("can`t allocate arrays, thus can`t deallocate them too"); |
|  | } |
|  | if (ptr == nullptr) { |
|  | return; |
|  | } |
|  | free\_blocks.push(reinterpret\_cast<char\*>(ptr)); |
|  | } |
|  |  |
|  | } |

1. **Результаты выполнения тестов**

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1

1 0

0 0

0 0

0 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

3. Добавить фигуру по индексу

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1

2 0

0 0

0 0

0 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

3. Добавить фигуру по индексу

2

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1

1 2

2 3

3 4

5 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

3. Добавить фигуру по индексу

3

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [1, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [1, 2] [2, 3] [3, 4] [5, 0]

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1

3 0

0 0

0 0

0 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

3. Добавить фигуру по индексу

2

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [1, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [1, 2] [2, 3] [3, 4] [5, 0]

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

5

0

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

2

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

31

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [1, 2] [2, 3] [3, 4] [5, 0]

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

3

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [1, 2] [2, 3] [3, 4] [5, 0]

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [2, 0] [0, 0] [0, 0] [0, 0]

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

2

1. Удалить фигуру в начале списка

2. Удалить фигуру в конце списка

3. Удалить фигуру по индексу

3

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

4

Trapeze: [3, 0] [0, 0] [0, 0] [0, 0]

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести кол-во фигур чья площаль больше чем ...

1. **Объяснение результатов работы программы**

Программа выводит меню, в котором описываются все применимые к фигурам функции – вставка, удаление и вывод фигур из трех различных мест, а также подсчет фигур с площадью большей чем заданное число. Функционально программа не изменилась, однако для реализованного ранее списка был написан аллокатор, который более грамотно распоряжается памятью, отведенной для хранения списка фигур.

1. **Вывод**

С помощью пользовательских аллокаторов программист может более эффективно распоряжаться отданной для хранения фигур памятью, сам следить за процессом выделения и очистки памяти, конструирования и деконструирования объектов.