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 5: «Основные работы с коллекциями: итераторы»

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №12 |
| Студент: | Коростелев Дмитрий Васильевич |
| Преподаватель: | Журавлёв Андрей Андреевич |
| Оценка: |  |
| Дата: | 25.11.2019 |

Москва, 2019

1. **Задание**

Разработать шаблоны классов согласно варианту задания. Параметром шаблона должен являться скалярный тип данных задающий тип данных для оси координат. Классы должны иметь публичные поля. Фигуры являются фигурами вращения. Для хранения координат фигур необходимо использовать шаблон std::pair.

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. Реализовать forward\_iterator по коллекции;

4. Коллекция должны возвращать итераторы begin() и end();

5. Коллекция должна содержать метод вставки на позицию итератора insert(iterator);

6. Коллекция должна содержать метод удаления из позиции итератора erase(iterator);

7. При выполнении недопустимых операций (например выход аз границы коллекции или удаление не

существующего элемента) необходимо генерировать исключения;

8. Итератор должен быть совместим со стандартными алгоритмами (например, std::count\_if)

9. Коллекция должна содержать метод доступа:

Стек – pop, push, top;

Очередь – pop, push, top;

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

10. Реализовать программу, которая:

Позволяет вводить с клавиатуры фигуры (с типом int в качестве параметра шаблона фигуры) и добавлять в коллекцию;

Позволяет удалять элемент из коллекции по номеру элемента;

Выводит на экран введенные фигуры c помощью std::for\_each;

Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if);

*Вариант 12: трапеция, список*

1. **Адрес репозитория на GitHub**

[https://github.com/Dmitry4K/oop\_exercise\_](https://github.com/Dmitry4K/oop_exercise_2)05

1. **Код программы на С++**

*main.cpp*

|  |
| --- |
| #include<iostream> |
|  | #include<algorithm> |
|  | #include<clocale> |
|  | #include"trapeze.h" |
|  | #include"containers/list.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 площадь 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>> MyList; |
|  |  |
|  | Trapeze<int> TempTrapeze; |
|  |  |
|  | 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::cout << MyList.length() << std::endl; |
|  | 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

|  |
| --- |
| #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() = default; |
|  | 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'; |
|  | return out; |
|  | } |
|  |  |
|  | template<class T> |
|  | std::istream& operator>> (std::istream& in, const Trapeze<T>& point) { |
|  | for (int i = 0; i < 4; i++) |
|  | in >> point.Vertexs[i]; |
|  | return in; |
|  | } |

*list.h*

|  |
| --- |
| #pragma once |
|  | #include <iterator> |
|  | #include <memory> |
|  |  |
|  |  |
|  | namespace containers { |
|  |  |
|  | template<class T> |
|  | class list { |
|  | private: |
|  | struct element;//îáúÿâëåíèå òèïà õðàíÿùåãîñÿ â list, äëÿ òîãî, ÷òîáû îí áûë âèäåí forward\_iterator |
|  | size\_t size = 0;//ðàçìåð ñïèñêà |
|  | public: |
|  | list() = default;//êîñòðóêòîð ïî óìîë÷àíèþ |
|  |  |
|  | 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: |
|  | struct element { |
|  | T value; |
|  | std::unique\_ptr<element> next\_element; |
|  | element\* prev\_element = nullptr; |
|  | element(const T& value\_) : value(value\_) {} |
|  | forward\_iterator next(); |
|  | }; |
|  |  |
|  | std::unique\_ptr<element> first; |
|  | element\* tail = nullptr; |
|  | }; |
|  |  |
|  | template<class T> |
|  | typename list<T>::forward\_iterator list<T>::begin() {//+ |
|  | return forward\_iterator(first.get()); |
|  | } |
|  |  |
|  | template<class T> |
|  | typename list<T>::forward\_iterator list<T>::end() {//+ |
|  | return forward\_iterator(nullptr); |
|  | } |
|  | template<class T> |
|  | size\_t list<T>::length() {//+ |
|  | return size; |
|  | } |
|  | template<class T> |
|  | bool list<T>::empty() { |
|  | return length() == 0; |
|  | } |
|  |  |
|  | template<class T> |
|  | void list<T>::push\_back(const T& value) { |
|  | //element\* result = new element(value); |
|  | if (!size) { |
|  | first = std::make\_unique<element>(value); |
|  | tail = first.get(); |
|  | size++; |
|  | return; |
|  | } |
|  | tail->next\_element = std::make\_unique<element>(value); |
|  | 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> |
|  | void list<T>::push\_front(const T& value) { |
|  | size++; |
|  | //element\* result = new element(value); |
|  | std::unique\_ptr<element> tmp = std::move(first); |
|  | first = std::make\_unique<element>(value); |
|  | first->next\_element = std::move(tmp); |
|  | if (first->next\_element != nullptr) |
|  | first->next\_element->prev\_element = first.get(); |
|  | if (size == 1) { |
|  | tail = first.get(); |
|  | } |
|  | if (size == 2) { |
|  | tail = first->next\_element.get(); |
|  | } |
|  | } |
|  |  |
|  | template<class T> |
|  | void list<T>::pop\_front() { |
|  | if (size == 0) { |
|  | throw std::logic\_error("can`t pop from empty list"); |
|  | } |
|  | if (size == 1) { |
|  | first = nullptr; |
|  | tail = nullptr; |
|  | size--; |
|  | return; |
|  | } |
|  | std::unique\_ptr<element> tmp = std::move(first->next\_element); |
|  | first = std::move(tmp); |
|  | first->prev\_element = nullptr; |
|  | size--; |
|  | } |
|  |  |
|  | template<class T> |
|  | void list<T>::pop\_back() { |
|  | if (size == 0) { |
|  | throw std::logic\_error("can`t pop from empty list"); |
|  | } |
|  | //tmp = nullptr; |
|  | if (tail->prev\_element) { |
|  | element\* tmp = tail->prev\_element; |
|  | //unique\_ptr dump = std::move(tail->prev\_element->next\_element); |
|  | tail->prev\_element->next\_element = nullptr; |
|  | tail = tmp; |
|  | //size--; |
|  | } |
|  | else { |
|  | first = nullptr; |
|  | tail = nullptr; |
|  | //tmp = first.get(); |
|  | //unique\_ptr dump = std::move(first); |
|  | } |
|  | //tail = tmp; |
|  | //first = pop\_impl(std::move(first)); |
|  | size--; |
|  | } |
|  |  |
|  |  |
|  | template<class T> |
|  | T& list<T>::front() { |
|  | if (size == 0) { |
|  | throw std::logic\_error("list is empty"); |
|  | } |
|  | return first->value; |
|  | } |
|  |  |
|  | template<class T> |
|  | T& list<T>::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> |
|  | list<T>& list<T>::operator=(list<T>& other) { |
|  | size = other.size; |
|  | first = std::move(other.first); |
|  | } |
|  |  |
|  | template<class T> |
|  | void list<T>::delete\_by\_it(containers::list<T>::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; |
|  | std::unique\_ptr<element> temp1 = std::move(d\_it.it\_ptr->next\_element); |
|  | d\_it.it\_ptr = d\_it.it\_ptr->prev\_element; |
|  | d\_it.it\_ptr->next\_element = std::move(temp1); |
|  | 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> |
|  | void list<T>::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> |
|  | void list<T>::insert\_by\_it(containers::list<T>::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); |
|  | \*/ |
|  | std::unique\_ptr<element> tmp = std::make\_unique<element>(value); |
|  | forward\_iterator i = this->begin(); |
|  | if (ins\_it == this->begin()) { |
|  | this->push\_front(value); |
|  | return; |
|  | } |
|  | if (/\*ins\_it.it\_ptr == tail || \*/ins\_it.it\_ptr == nullptr) { |
|  | this->push\_back(value); |
|  | return; |
|  | } |
|  |  |
|  | tmp->prev\_element = ins\_it.it\_ptr->prev\_element; |
|  | ins\_it.it\_ptr->prev\_element = tmp.get(); |
|  | tmp->next\_element = std::move(tmp->prev\_element->next\_element); |
|  | //tmp->prev\_element->next\_element = unique\_ptr(tmp, deleter{ &this->allocator\_ }); |
|  | tmp->prev\_element->next\_element = std::move(tmp); |
|  |  |
|  | size++; |
|  | } |
|  |  |
|  | template<class T> |
|  | void list<T>::insert\_by\_number(size\_t N, T& value) { |
|  | forward\_iterator it = this->begin(); |
|  | if (N >= this->length()) |
|  | it = this->end(); |
|  | else |
|  | for (size\_t i = 0; i < N; ++i) { |
|  | ++it; |
|  | } |
|  | this->insert\_by\_it(it, value); |
|  | } |
|  | template<class T> |
|  | typename list<T>::forward\_iterator list<T>::element::next() { |
|  | return forward\_iterator(this->next\_element.get()); |
|  | } |
|  |  |
|  | template<class T> |
|  | list<T>::forward\_iterator::forward\_iterator(containers::list<T>::element\* ptr) { |
|  | it\_ptr = ptr; |
|  | } |
|  |  |
|  | template<class T> |
|  | T& list<T>::forward\_iterator::operator\*() { |
|  | return this->it\_ptr->value; |
|  | } |
|  | template<class T> |
|  | T& list<T>::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> |
|  | typename list<T>::forward\_iterator& list<T>::forward\_iterator::operator++() { |
|  | if (it\_ptr == nullptr) throw std::logic\_error("out of list borders"); |
|  | \*this = it\_ptr->next(); |
|  | return \*this; |
|  | } |
|  |  |
|  | template<class T> |
|  | typename list<T>::forward\_iterator list<T>::forward\_iterator::operator++(int) { |
|  | forward\_iterator old = \*this; |
|  | ++\* this; |
|  | return old; |
|  | } |
|  |  |
|  | template<class T> |
|  | bool list<T>::forward\_iterator::operator==(const forward\_iterator& other) const { |
|  | return it\_ptr == other.it\_ptr; |
|  | } |
|  |  |
|  | template<class T> |
|  | bool list<T>::forward\_iterator::operator!=(const forward\_iterator& other) const { |
|  | return it\_ptr != other.it\_ptr; |
|  | } |
|  | } |

1. **Результаты выполнения тестов**

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

1

0 0

1 1

2 2

3 3

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

1

0 0 0 0 0 0 0 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

1

1 1 1 1 1 1 1 1

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

2

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

4

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

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

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

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

3

1. Вывести первую фигуру в списке

2. Вывести последнюю фигуру в списке

3. Вывести фигуру по индексу

1

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

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

3

1. Вывести первую фигуру в списке

2. Вывести последнюю фигуру в списке

3. Вывести фигуру по индексу

2

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

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

3

1. Вывести первую фигуру в списке

2. Вывести последнюю фигуру в списке

3. Вывести фигуру по индексу

3

1

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

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь 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 фигуру если площадь больше чем ...

4

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

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

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь 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 фигуру если площадь больше чем ...

4

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

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

1

0 0 0 0 0 0 0 0

1. Добавить фигуру в начало списка

2. Добавить фигуру в конец списка

1

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

3

1. Вывести первую фигуру в списке

2. Вывести последнюю фигуру в списке

3. Вывести фигуру по индексу

3

0

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

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

5

0

0

1. Добавить фигуру в список

2. Удалить фигуру

3. Вывести фигуру

4. Вывести все фигуры

5. Вывести фигуру если площадь больше чем ...

1. **Объяснение результатов работы программы**

Программа выводит диалоговое окно, в котором представлен полный функционал работы с однонаправленным списком. Все элементы списка связаны между собой с помощью unque\_ptr, а взаимодействие со списком происходит через итераторы. В реализованном мной списке пользователь может, добавлять элементы в начало/конец/по итератору новые элементы, печатать первый/последний/по индексу элементы, а также удалять элементы в начале/конце/по номеру/по итератору

1. **Вывод**

Благодаря итераторам, при их грамотной настройке программист получает более наглядный и простой способ работы с контейнерами и другими абстрактными типами данных, кроме того, правильная реализация итераторов в собственном типе данных дает программисту возможность использования уже написанных алгоритмов, в основе которых лежит взаимодействие через итераторы.