**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 5**

Тема: Основы работы с коллекциями: итераторы

Студент: Чернобаев Андрей Александрович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

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, т.е. равносторонними (кроме трапеции и прямоугольника). Для хранения координат фигур необходимо использовать шаблон std::pair.

Например:

template <class T>

struct Square{

using vertex\_t = std::pair<T,T>;

vertex\_t a,b,c,d;

};

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 - фигуры;

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:

o Стек – pop, push, top;

o Очередь – pop, push, top;

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

10. Реализовать программу, которая:

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

o Позволяет удалять элемент из коллекции по номеру элемента;

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

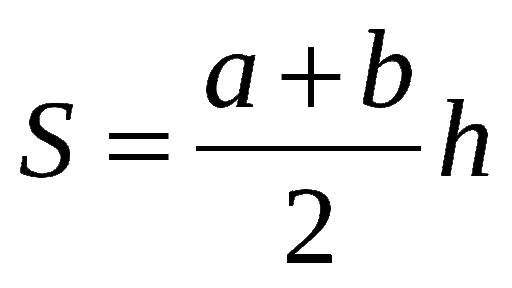
o Выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count\_if);

**Варианты заданий**

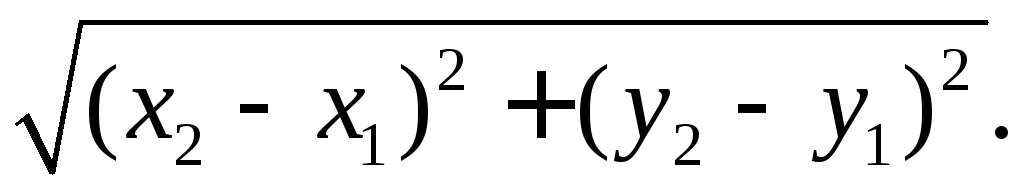
|  |  |  |
| --- | --- | --- |
| **Вариант** | **Фигура** | **Контейнер** |
| 12 | Трапеция | Список |

1. Описание программы

Формула площади трапеции:



Формула расстояние между двумя точками:



Программа состоит из 8-ми файлов.

accessory.h - Заголовочный файл, содержит дополнительные функции программы, облегчающие ввод-вывод.

Состоит из:

* 3-х лямбдо-функций.
* 2-ух функций, реализующих split.
* функция help.

List.h - заголовочный файл для реализации списка согласно заданию

Состоит из:

* класса List
* класса forward\_iterator
* класса Element

Trapezoid.h - заголовочный файл для трапеции.

Состоит из:

* класса Trapezoid
* класса перегрузки оператора вывода на печать для класса Trapezoid

List.cpp - реализация функций класса List из List.h

forward\_iterator.cpp - реализация класса forward\_iterator из List.h

Element.cpp - реализация класса Element из List.h

1. Набор тестов

**Из файла test\_01.txt:**

0 0 4 4 9 4 14 0

0 0 4 4 9 4 13 0

0 0 4 4 9 4 12 0

delete 1

show

delete 0

show

0 0 2 4 9 4 11 0

show

less 200

less 36

exit

**Из файла test\_02.txt:**

5 5 4 12 5 12 20 5

show

less 300

4 5 1 12 6 12 21 5

show

less 20

delete 0

show

0 0 4 4 9 4 13 0

0 0 4 4 9 4 12 0

5 5 4 12 6 12 20 5

delete 2

show

exit

1. Результаты выполнения тестов

|  |  |
| --- | --- |
| input | output |
| 0 0 4 4 9 4 14 0  0 0 4 4 9 4 13 0  0 0 4 4 9 4 12 0  delete 1  show  delete 0  show  0 0 2 4 9 4 11 0  show  less 200  less 36  exit | write exit for exit  write help for help  write delete {index} to remove figure by index  write show to show all figures  write less {square} to show figures with square less, than {square}  input format x\_1 y\_1 x\_2 y\_2 ... x\_n y\_n figure  begin of Trapezoid  0 0  4 4  9 4  14 0  Square is: 38  end of Trapezoid  begin of Trapezoid  0 0  4 4  9 4  12 0  Square is: 34  end of Trapezoid  begin of Trapezoid  0 0  4 4  9 4  12 0  Square is: 34  end of Trapezoid  begin of Trapezoid  0 0  4 4  9 4  12 0  Square is: 34  end of Trapezoid  begin of Trapezoid  0 0  2 4  9 4  11 0  Square is: 36  end of Trapezoid  2  1 |
| 5 5 4 12 5 12 20 5  show  less 300  4 5 1 12 6 12 21 5  show  less 20  delete 0  show  0 0 4 4 9 4 13 0  0 0 4 4 9 4 12 0  5 5 4 12 6 12 20 5  delete 2  show  exit | write exit for exit  write help for help  write delete {index} to remove figure by index  write show to show all figures  write less {square} to show figures with square less, than {square}  input format x\_1 y\_1 x\_2 y\_2 ... x\_n y\_n figure  begin of Trapezoid  5 5  4 12  5 12  20 5  Square is: 56  end of Trapezoid  1  begin of Trapezoid  5 5  4 12  5 12  20 5  Square is: 56  end of Trapezoid  begin of Trapezoid  4 5  1 12  6 12  21 5  Square is: 77  end of Trapezoid  0  begin of Trapezoid  4 5  1 12  6 12  21 5  Square is: 77  end of Trapezoid  begin of Trapezoid  4 5  1 12  6 12  21 5  Square is: 77  end of Trapezoid  begin of Trapezoid  0 0  4 4  9 4  13 0  Square is: 36  end of Trapezoid  begin of Trapezoid  5 5  4 12  6 12  20 5  Square is: 59.5  end of Trapezoid |

1. Листинг программы

https://github.com/ruthenorum/oop\_exercise\_05

Из файла **main.cpp**

#include <iostream>

#include <algorithm>

#include "List.h"

#include "Trapezoid.h"

#include "accessory.h"

using f\_type = int;

int main() {

std::string cmd;

std::vector<std::string> out;

std::stringstream buffer;

List<Trapezoid<f\_type>> list;

help();

while (true){

getline(std::cin, cmd);

std::vector<std::string> cmd\_array = split(cmd,' ');

if (cmd\_array[0] == "exit"){

break;

}

else if (cmd\_array[0] == "help"){

help();

}

else if (cmd\_array[0] == "delete"){

size\_t index = string\_to\_int(cmd\_array[1]);

list.erase\_by\_index(index);

}

else if (cmd\_array[0] == "show"){

std::for\_each(list.begin(), list.end(), [](Trapezoid<f\_type>& T) { std::cout << T; });

}

else if (cmd\_array[0] == "less"){

double S = string\_to\_double(cmd\_array[1]);

std::cout <<

std::count\_if(list.begin(), list.end(), [=](Trapezoid<f\_type>& T) { return T.get\_square() < S; })

<< std::endl;

}

else {

std::vector<std::string> array = split(cmd,' ');

std::vector<std::pair<f\_type,f\_type>> numbers\_double(EDGES\_COUNT);

int j = 0;

for(size\_t i = 0;i < numbers\_double.size() \* 2 ; i += 2){

numbers\_double[j] = {string\_to\_double(array[i]),string\_to\_double(array[i+1])};

j++;

}

Trapezoid<f\_type> tr(numbers\_double);

list.insert\_by\_index(list.length(),tr);

}

}

return 0;

}

Из файла **accessory.h**

#ifndef TASK5\_ACCESSORY\_H

#define TASK5\_ACCESSORY\_H

#include <string>

#include <vector>

#include <sstream>

auto string\_to\_int = [](const std::string& s) {return std::stoi(s);};

auto string\_to\_double = [](const std::string& s) {return std::stod(s);};

auto int\_to\_string = [](const int i) {return std::to\_string(i);};

template <typename Out>

void split(const std::string &s, char delim, Out result) {

std::istringstream iss(s);

std::string item;

while (std::getline(iss, item, delim)) {

\*result++ = item;

}

}

std::vector<std::string> split(const std::string &s, char delim) {

std::vector<std::string> elems;

split(s, delim, std::back\_inserter(elems));

return elems;

}

void help(){

std::cout << "write exit for exit" << std::endl;

std::cout << "write help for help" << std::endl;

std::cout << "write delete {index} to remove figure by index" << std::endl;

std::cout << "write show to show all figures" << std::endl;

std::cout << "write less {square} to show figures with square less, than {square}" << std::endl;

std::cout << "input format x\_1 y\_1 x\_2 y\_2 ... x\_n y\_n figure" << std::endl;

std::cout << "" << std::endl;

}

#endif //TASK5\_ACCESSORY\_H

Из файла **Element.cpp**

template <class T>

forward\_iterator<T> Element<T>::next(){

return forward\_iterator<T>(next\_el.get());

}

Из файла forward\_iterator.cpp

template <class T>

forward\_iterator<T>::forward\_iterator(Element<T>\* \_ptr){

ptr = \_ptr;

}

template<class T>

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

return ptr->value;

}

template<class T>

forward\_iterator<T>& forward\_iterator<T>::forward\_iterator::operator++() {

if (ptr == nullptr){

throw std::out\_of\_range("out of range");

}

\*this = ptr->next();

return \*this;

}

template<class T>

forward\_iterator<T> forward\_iterator<T>::forward\_iterator::operator++(int) {

forward\_iterator<T> old = \*this;

++\*this;

return old;

}

template<class T>

bool forward\_iterator<T>::forward\_iterator::operator==(const forward\_iterator& Other) const {

return ptr == Other.ptr;

}

template<class T>

bool forward\_iterator<T>::forward\_iterator::operator!=(const forward\_iterator& Other) const {

return ptr != Other.ptr;

}

Из файла **List.cpp**

template<class T>

std::shared\_ptr<Element<T>> pop(std::shared\_ptr<Element<T>> el) {

if (el->next\_el != nullptr) {

el->next\_el = pop(el->next\_el);

return el;

}

return nullptr;

}

template<class T>

std::shared\_ptr<Element<T>> push(std::shared\_ptr<Element<T>> el) {

if (el -> next\_el != nullptr) {

return push(el->next\_el);

}

return el;

}

template <class T>

const size\_t List<T>::length() const{

return size;

}

template <class T>

forward\_iterator<T> List<T>::begin() {

return forward\_iterator<T>(first.get());

}

template <class T>

forward\_iterator<T> List<T>::end() {

return forward\_iterator<T>(nullptr);

}

template <class T>

void List<T>::erase\_by\_it(forward\_iterator<T> it){

if (size == 0){

throw std::invalid\_argument("no elements");

}

if (it.ptr == nullptr){

throw std::invalid\_argument("no element");

}

if (it == begin()){

first = first->next\_el;

if (first != nullptr){

first->prev\_el.reset();

}

size--;

return;

}

if (it == end()){

first = pop(first);

size--;

return;

}

std::shared\_ptr<Element<T>> after\_deleted = it.ptr->next\_el;

std::weak\_ptr<Element<T>> before\_deleted = it.ptr->prev\_el;

after\_deleted->prev\_el = before\_deleted;

before\_deleted.lock()->next\_el = after\_deleted;

size--;

}

template<class T>

void List<T>::erase\_by\_index(const size\_t index) {

forward\_iterator<T> it = begin();

for (size\_t i = 1; i <= index; ++i) {

it++;

}

erase\_by\_it(it);

}

template<class T>

void List<T>::insert\_by\_it(forward\_iterator<T> it,const T& value){

if (first == nullptr){

first = std::shared\_ptr<Element<T>>(new Element<T>{value});

}

else {

auto temp = std::shared\_ptr<Element<T>>(new Element<T>{value});

if (it == begin()){

temp->next\_el = first;

first->prev\_el = temp;

first = temp;

}

else if (it.ptr == nullptr){

temp->prev\_el = push(first);

push(first)->next\_el = temp;

}

else {

temp->prev\_el = it.ptr->prev\_el;

temp->next\_el = it.ptr->prev\_el.lock()->next\_el;

it.ptr->prev\_el = temp;

temp->prev\_el.lock()->next\_el = temp;

}

}

size++;

}

template <class T>

void List<T>::insert\_by\_index(const size\_t n,const T& value){

forward\_iterator<T> it = begin();

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

it++;

}

insert\_by\_it(it, value);

}

template<class T>

List<T>& List<T>::operator=(List<T>&& Other){

size = Other.size;

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

}

template<class T>

T& List<T>::operator[](const size\_t index) const {

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

throw std::out\_of\_range("out of range");

}

forward\_iterator<T> it = begin();

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

it++;

}

return \*it;

}

Из файла **List.h**

#ifndef OOP5\_LIST\_H

#define OOP5\_LIST\_H

#include <memory>

template <class T>

class forward\_iterator;

template <class T>

class Element;

template <class T>

class List {

public:

List() = default;

const size\_t length() const;

forward\_iterator<T> begin();

forward\_iterator<T> end();

void erase\_by\_it(forward\_iterator<T> it);

void erase\_by\_index(const size\_t n);

void insert\_by\_it(forward\_iterator<T> it,const T& value);

void insert\_by\_index(const size\_t n,const T& value);

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

List<T>& operator=(List<T>&& Other);

private:

size\_t size = 0;

std::shared\_ptr<Element<T>> first = nullptr;

};

template <class T>

class forward\_iterator {

public:

using value\_type = T;

using reference = T&;

using pointer = T\*;

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

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

explicit forward\_iterator(Element<T>\* 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<T>\* ptr;

friend List<T>;

};

template <class T>

class Element {

public:

T value;

std::shared\_ptr<Element> next\_el;

std::weak\_ptr<Element> prev\_el ;

forward\_iterator<T> next();

};

#include "List.cpp"

#include "Element.cpp"

#include "forward\_iterator.cpp"

#endif //OOP5\_LIST\_H

Из файла **Trapezoid.cpp**

#include "Trapezoid.h"

double get\_distance(const std::pair<double,double>& one,const std::pair<double,double>& two){

double x\_diff = (two.first - one.first);

double y\_diff = (two.second - one.second);

return sqrt(x\_diff \* x\_diff + y\_diff \* y\_diff);

}

template<class T>

double Trapezoid<T>::get\_square() const{

double less\_base = get\_distance(edges[1],edges[2]);

double larger\_base = get\_distance(edges[0],edges[3]);

int x1 = edges[0].first;

int x2 = edges[3].first;

int y1 = edges[0].second;

int y2 = edges[3].second;

int x0 = edges[1].first;

int y0 = edges[1].second;

double numerator = fabs((y2 - y1) \* x0 - (x2 - x1)\*y0 + x2\*y1 - y2\*x1);

double denominator = sqrt((y2-y1)\*(y2-y1) + (x2-x1)\*(x2-x1));

double height = numerator / denominator;

return 0.5 \* (less\_base + larger\_base) \* height;

}

template<class T>

Trapezoid<T>::Trapezoid(vertex\_t& \_a,vertex\_t& \_b,vertex\_t& \_c,vertex\_t& \_d) {

a = \_a;

b = \_b;

c = \_c;

d = \_d;

edges.resize(EDGES\_COUNT);

edges[0].first = static\_cast<double>(a.first);

edges[0].second = static\_cast<double>(a.second);

edges[1].first = static\_cast<double>(b.first);

edges[1].second = static\_cast<double>(b.second);

edges[2].first = static\_cast<double>(c.first);

edges[2].second = static\_cast<double>(c.second);

edges[3].first = static\_cast<double>(d.first);

edges[3].second = static\_cast<double>(d.second);

}

template<class T>

Trapezoid<T>::Trapezoid(std::vector<std::pair<T, T>> \_vec) {

a = \_vec[0];

b = \_vec[1];

c = \_vec[2];

d = \_vec[3];

edges.resize(EDGES\_COUNT);

edges[0].first = static\_cast<double>(a.first);

edges[0].second = static\_cast<double>(a.second);

edges[1].first = static\_cast<double>(b.first);

edges[1].second = static\_cast<double>(b.second);

edges[2].first = static\_cast<double>(c.first);

edges[2].second = static\_cast<double>(c.second);

edges[3].first = static\_cast<double>(d.first);

edges[3].second = static\_cast<double>(d.second);

}

std::ostream& operator << (std::ostream& out, const std::pair<double,double>& p){

out << p.first << " " << p.second << std::endl;

return out;

}

template <class T>

std::ostream &operator<<(std::ostream& out, const Trapezoid<T>& tr) {

out << "begin of Trapezoid" << std::endl;

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

out << tr.edges[i];

}

out << "Square is: " << tr.get\_square() << std::endl;

out << "end of Trapezoid" << std::endl;

return out;

}

Из файла **Trapezoid.h**

#ifndef OOP5\_TRAPEZOID\_H

#define OOP5\_TRAPEZOID\_H

#include <utility>

#include <cmath>

#include <vector>

const int EDGES\_COUNT = 4;

template <class T>

class Trapezoid {

public:

using vertex\_t = std::pair<T,T>;

vertex\_t a,b,c,d;

std::vector<std::pair<double,double>> edges;

double get\_square() const;

explicit Trapezoid(vertex\_t& a,vertex\_t& b,vertex\_t& c,vertex\_t& d);

explicit Trapezoid(std::vector<std::pair<T,T>> \_vec);

};

template <class T>

std::ostream& operator << (std::ostream& out, const Trapezoid<T>& tr);

#include "Trapezoid.cpp"

#endif //OOP5\_TRAPEZOID\_H

1. Выводы.

Я научился реализовывать коллекции на умных указателях и работать с итераторами в C++.

1. Список использованных источников.

1 std::shared\_ptr [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/memory/shared\_ptr

2 Templates [Электронный ресурс].

URL: https://www.cplusplus.com/doc/oldtutorial/templates/

3 std::tuple [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/utility/tuple

4 std::weak\_ptr [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/memory/weak\_ptr

5 static\_cast conversion [Электронный ресурс].

URL: https://en.cppreference.com/w/cpp/language/static\_cast