**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: Воронов Кирилл Михайлович

Группа: 80-207

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.

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 - 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 за границы коллекции или удаление несуществующего 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 27 прямоугольник, динамический массив*

1. **Описание программы**

В файле vector.hpp реализован класс TVector с переменными Max(размером вектора) и умным указателем shared\_ptr<T> Data. Также в нём находится класс Iterator, который тоже имеет умный указатель shared\_ptr<T> ptr, переменные size(размер вектора) и index(текущий индекс вектора, на который указывает итератор). Для итератора и вектора реализованы соответствующие операторы и методы (добавление через итератор insert, удаление erase). В main.cpp реализован класс Rectangle с парой t для координат и переменными a,b - длинами сторон прямоугольника. Взаимодействие с программой происходит через интерактивное меню, в котором указаны номера команд и их действие. Команды вызывают соответствующие функции.

1. **Набор тестов и результаты их выполнения**

Введите 1, чтобы добавить элемент

Введите 2, чтобы удалить элемент

Введите 3, чтобы вывести вектор

Введите 4, чтобы вывести фигуры меньшей заданной площади

Введите 5, чтобы вывести меню

1

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

1 1 1 1

1

Введите индекс элемента массива, перед которым добавить элемент

1

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

0 0 1.56 3.23

3

(0,3.23) (1.56,3.23)

(0,0) (1.56,0)

-----------------

(1,2) (2,2)

(1,1) (2,1)

-----------------

1

Введите индекс элемента массива, перед которым добавить элемент

2

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

0 0 200 145

1

Введите индекс элемента массива, перед которым добавить элемент

3

Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной

9 9 665 2.365

3

(0,3.23) (1.56,3.23)

(0,0) (1.56,0)

-----------------

(1,2) (2,2)

(1,1) (2,1)

-----------------

(0,145) (200,145)

(0,0) (200,0)

-----------------

(9,11.365) (674,11.365)

(9,9) (674,9)

-----------------

5

Введите 1, чтобы добавить элемент

Введите 2, чтобы удалить элемент

Введите 3, чтобы вывести вектор

Введите 4, чтобы вывести количество фигур, меньших заданной площади

Введите 5, чтобы вывести меню

4

Введите площадь

5000

3

2

Введите индекс элемента для удаления

0

2

Введите индекс элемента для удаления

0

2

Введите индекс элемента для удаления

0

2

Введите индекс элемента для удаления

0

3

Вектор пуст

1. **Листинг программы**

vector.hpp

#ifndef VECTOR\_HPP

#define VECTOR\_HPP

#include<iostream>

#include<memory>

using namespace std;

template <class T>

class TVector {

public:

int Max = 0;

shared\_ptr<T[]> Data = nullptr;

class Iterator {

public:

typedef Iterator self;

typedef T value\_type;

typedef T &reference;

typedef T \*pointer;

typedef std::random\_access\_iterator\_tag iterator\_category;

typedef ptrdiff\_t difference\_type;

shared\_ptr<T[]> ptr;

int size;

int index = 0;

public:

Iterator(shared\_ptr<T[]> ptr, size\_t const index, int size) : ptr(ptr), index(index), size(size) {};

Iterator(){

}

T operator\*(){

return ptr[index];

}

self &operator++() {

if (index >= size)

throw std::out\_of\_range("Итератор вышел за пределы вектора\n");

index+=1;

return \*this;

}

bool operator<(const Iterator &b) {

return index < b.index;

}

bool operator!=(const Iterator &b) {

return index != b.index;

}

};

Iterator begin(){

return Iterator(Data, 0, Max);

}

Iterator end(){

return Iterator(Data, Max, Max);

}

void insert(T z, Iterator it) {

if(Max == 0) {

shared\_ptr<T[]> Data2{new T[1]};

Data = Data2;

Data[0] = z;

Max = 1;

it.ptr = Data;

it.size = 1;

} else {

shared\_ptr<T[]> Data2{new T[Max + 1]};

for (int i = 0; i < it.index; ++i){

Data2[i] = Data[i];

}

Data2[it.index] = z;

for (int i = it.index + 1; i < Max + 1; ++i){

Data2[i] = Data[i-1];

}

it.ptr = Data2;

it.size+=1;

Data = Data2;

Max+=1;

}

}

void erase(Iterator it) {

if (Max == 1) {

Data = nullptr;

Max = 0;

it.ptr = nullptr;

it.size = 0;

it.index = 0;

} else {

shared\_ptr<T[]> Data2{new T[Max - 1]};

for (int i = 0; i < it.index; ++i) {

Data2[i] = Data[i];

}

for (int i = it.index + 1; i < Max; ++i){

Data2[i-1] = Data[i];

}

Data = Data2;

Max-=1;

it.ptr = Data;

it.size-=1;

}

}

int size() {

return Max;

}

T& operator[](int i){

return Data[i];

}

};

#endif

main.cpp

/\* Воронов К.М.

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.

Создать шаблон динамической коллекцию и итератор.

\*/

#include <iostream>

#include <string>

#include <algorithm>

#include"vector.hpp"

template <typename T>

class Rectangle{

public:

using type = T;

pair<T,T> t;

T a;

T b;

Rectangle(){};

Rectangle(T x,T y,T a1, T b1): t(x,y), a(a1), b(b1){};

};

double square(Rectangle<double> p) {

return p.a\*p.b;

}

void print2(Rectangle<double> p) {

pair<double,double> t2((p.t).first + p.a, (p.t).second );

pair<double,double> t3((p.t).first + p.a, (p.t).second + p.b);

pair<double,double> t4((p.t).first, (p.t).second + p.b);

cout << "(" << t4.first << "," << t4.second << ") "

<< "(" << t3.first << "," << t3.second << ")"

<< endl << endl << "(" << (p.t).first << ","

<< (p.t).second << ") " << "(" << t2.first << ","

<< t2.second << ")" << endl << "-----------------" << endl;

}

void Add(TVector<Rectangle<double>> &v, TVector<Rectangle<double>>::Iterator &begin, Rectangle<double> r){

double x,y,a,b;

if (v.size() < 1) {

printf("Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной\n");

cin >> x >> y >> a >> b;

if ((a<0) || (b<0) || !cin){

cout << "Введены неверные данные" << endl;

return;

}

(r.t).first = x;

(r.t).second = y;

r.a = a;

r.b = b;

v.insert(r, begin);

return;

}

int index;

printf("Введите индекс элемента массива, перед которым добавить элемент\n");

cin >> index;

begin = v.begin();

if(index > v.size() || index < 0 || !cin){

cout << "Введены неверные данные" << endl;

return;

}

while(begin.index < index) {

++begin;

}

printf("Введите координату нижней левой точки прямоугольника и длины сторон, прилегающих к ней, начиная с горизонтальной\n");

cin >> x >> y >> a >> b;

if ((a<0) || (b<0) || !cin){

cout << "Введены неверные данные" << endl;

return;

}

(r.t).first = x;

(r.t).second = y;

r.a = a;

r.b = b;

v.insert(r, begin);

return;

}

void Delete(TVector<Rectangle<double>> &v, TVector<Rectangle<double>>::Iterator &begin){

int index;

if (v.size() == 0){

cout <<"Вектор пуст" << endl;

return;

}

printf("Введите индекс элемента для удаления\n");

cin >> index;

if((index < 0) || (!cin) || (index > v.size())) {

cout << "Введены неверные данные" << endl;

return;

}

begin = v.begin();

while(begin.index < index) {

++begin;

}

v.erase(begin);

return;

}

double s;

bool ifsquare(Rectangle<double> p){

if (square(p) < s)

return true;

else return false;

}

void Square(TVector<Rectangle<double>> ::Iterator &it,TVector<Rectangle<double>> v ){

printf("Введите площадь\n");

cin >> s;

if(!cin){

cout << "Введены неверные данные" << endl;

return;

}

int a;

a = count\_if(v.begin(),v.end(), ifsquare);

cout << a << endl;

}

void Print(TVector<Rectangle<double>> ::Iterator &it,TVector<Rectangle<double>> v ){

if (v.size() == 0){

cout <<"Вектор пуст" << endl;

return;

}

for\_each(v.begin(), v.end(),print2);

return;

}

void Menu(){

printf("Введите 1, чтобы добавить элемент\n");

printf("Введите 2, чтобы удалить элемент\n");

printf("Введите 3, чтобы вывести вектор\n");

printf("Введите 4, чтобы вывести количество фигур, меньших заданной площади\n");

printf("Введите 5, чтобы вывести меню\n");

}

int main(){

TVector<Rectangle<double>> v;

auto begin = v.begin();

auto end = v.end();

double x,y,a,b;

int input;

Rectangle<double> r;

printf("Введите 1, чтобы добавить элемент\n");

printf("Введите 2, чтобы удалить элемент\n");

printf("Введите 3, чтобы вывести вектор\n");

printf("Введите 4, чтобы вывести фигуры меньшей заданной площади\n");

printf("Введите 5, чтобы вывести меню\n");

while (cin >> input) {

if (!cin){

cout << "Введены неверные данные" << endl;

return -1;

}

switch(input){

case 1:

Add(v,begin,r);

break;

case 2:

Delete(v,begin);

break;

case 3:

Print(begin,v);

break;

case 4:

Square(begin,v);

break;

case 5:

Menu();

break;

default:

cout<<"Вы ввели неверную команду"<<std::endl;

return -1;

break;

}

}

}

**Список литературы**

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

URL:https://en.cppreference.com/w/cpp/memory/shared\_ptr (дата обращения 14.11.2020).

1. Мануал std::for\_each [Электронный ресурс]

URL:https://en.cppreference.com/w/cpp/algorithm/for\_each (дата обращения 15.11.2020).