**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: Лисин Роман Сергеевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

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 (правильные многоугольники), 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:

* очередь – pop, push, top;

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 19: В качестве коллекции использую очередь, а фигура - прямоугольник.

1. **Описание программы**

Программа является интерактивной и сама выводит весь свой функционал для пользователя. В качестве умного указателя я использовал std::unique\_ptr.

1. **Набор тестов**

**2 1 2 3 4 4 3 0 4 0**

**2 1 2 3 4 2 5 8 10 20 4 5 50 0**

1. **Результаты выполнения тестов**

0 Выход из программы

1 Запрос меню

2 Добавить фигуру в очередь

3 Удалить фигуру из очереди по номеру

4 Печать фигур в очереди

5 Печать количества фигур, площадь которых меньше заданной

Введите команду: 2

Введите координаты положения прямоугольника: 1 2

Введите ширину прямоугольника: 3

Введите высоту прямоугольника: 4

Введите команду: 2

Введите координаты положения прямоугольника: 5 8

Введите ширину прямоугольника: 10

Введите высоту прямоугольника: 20

Введите команду: 4

Точки прямоугольника: (1, 2) (4, 2) (4, -2) (1, -2)

Точки прямоугольника: (5, 8) (15, 8) (15, -12) (5, -12)

Введите команду: 5

Введите площадь: 50

1

1. **Листинг программы**

rectangle.hpp

#include <iostream>

using namespace std;

template <class T>

struct Rectangle {

pair<T, T> point;

T height;

T width;

Rectangle() : point{ 0,0 }, height(0), width(0) {}

Rectangle(T x, T y, T h, T w) : point{ x,y } {

if (h < 0 or w < 0) {

throw "Ширина и высота должны быть положительными!";

}

height = h;

width = w;

}

T Area() {

return height \* width;

}

};

template <class T>

ostream& operator<< (ostream& out, const unique\_ptr<Rectangle<T>>& A) {

out << "Точки прямоугольника: " << "(" << A->point.first << ", " << A->point.second << ") "

<< "(" << A->point.first + A->width << ", " << A->point.second << ") "

<< "(" << A->point.first + A->width << ", " << A->point.second - A->height << ") "

<< "(" << A->point.first << ", " << A->point.second - A->height << ") "

<< "\n";

return out;

}

template <class T>

istream& operator>> (istream& in, unique\_ptr<Rectangle<T>>& A) {

cout << "Введите координаты положения прямоугольника: ";

in >> A->point.first >> A->point.second;

cout << "Введите ширину прямоугольника: ";

in >> A->width;

cout << "Введите высоту прямоугольника: ";

in >> A->height;

if ((A->width < 0) || (A->height < 0)) {

throw "Ширина и высота должны быть положительными!";

}

return in;

}

template <class T>

void print(unique\_ptr<Rectangle<T>>& elem) {

cout << elem;

}

queue.hpp

#include <iostream>

#include <memory> // unique\_ptr

using namespace std;

template <class T>

struct Queue\_iterator {

using figure = T;

using iterator\_category = forward\_iterator\_tag;

using value\_type = unique\_ptr<T>;

using pointer = unique\_ptr<T>\*;

using reference = unique\_ptr<T>&;

using difference\_type = unsigned int;

pointer it;

Queue\_iterator() {}

Queue\_iterator(pointer queue) : it(queue) {}

void next() {

it += 1;

}

reference operator\*() {

return \*it;

}

Queue\_iterator<figure>& operator++() {

it += 1;

return \*this;

}

Queue\_iterator<figure>& operator+(int a) {

it += a;

return \*this;

}

Queue\_iterator<figure>& operator-(int a) {

it -= a;

return \*this;

}

bool operator!=(const Queue\_iterator& rvl) {

return it != rvl.it;

}

};

template <class T>

struct Queue {

using figure = T;

using iterator = Queue\_iterator<T>;

using value\_type = unique\_ptr<T>;

using pointer = unique\_ptr<T>\*;

unsigned int size;

unsigned int buffer;

pointer queue;

Queue() : size(0), buffer(0), queue(NULL) {}

~Queue() {

delete[]queue;

}

void Push(value\_type elem) {

if (buffer == 0) {

queue = new value\_type[1];

buffer = 1;

}

if (size == buffer) {

value\_type\* temp = new value\_type[buffer << 1];

buffer <<= 1;

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

temp[i] = move(queue[i]);

}

delete[]queue;

queue = temp;

}

queue[size] = move(elem);

++size;

}

void Pop() {

if (size == 0) {

return;

}

if (size > 1) {

for (unsigned int i = 0; i < size - 1; ++i) {

queue[i] = move(queue[i + 1]);

}

}

--size;

if (size \* 2 < buffer) {

value\_type\* temp = new value\_type[buffer >> 1];

buffer >>= 1;

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

temp[i] = move(queue[i]);

}

delete[] queue;

queue = temp;

}

}

void Remove(unsigned int id) {

if (id >= size) {

throw "Индекс больше размера очереди!";

}

for (unsigned int i = 0; i < id; ++i) {

value\_type s = Top();

Pop();

Push(move(s));

}

Pop();

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

Push(Top());

Pop();

}

}

value\_type Top() {

if (size > 0) {

return make\_unique<figure>(\*queue[0]);

}

else {

throw "Очередь пуста!";

}

}

void insert(iterator it, value\_type elem) {

\*it = elem;

}

void erase(iterator it) {

while (it + 1 != end()) {

\*it = \*(it + 1);

it++;

}

size--;

}

iterator begin() {

return iterator(queue);

}

iterator end() {

return iterator(queue + size);

}

};

main.cpp

#include <iostream>

#include <memory> // unique\_ptr

#include <algorithm> // for\_each, count\_if

#include <clocale>

#include "rectangle.hpp"

#include "queue.hpp"

/\* Лабораторная работа 5. Выполнил Лисин Роман, группа М8О-206Б-20ю

Вариант 19: Очередь с прямоугольниками

\*/

using namespace std;

using figure\_type = int;

void writeMenu() {

cout << "0\tВыход из программы\n";

cout << "1\tЗапрос меню\n";

cout << "2\tДобавить фигуру в очередь\n";

cout << "3\tУдалить фигуру из очереди по номеру\n";

cout << "4\tПечать фигур в очереди\n";

cout << "5\tПечать количества фигур, площадь которых меньше заданной\n";

}

int main() {

setlocale(LC\_ALL, "rus");

Queue<Rectangle<figure\_type>> st;

short command;

bool menu = 1;

writeMenu();

while (menu) {

cout << '\n';

cout << "Введите команду: ";

cin >> command;

cout << '\n';

switch (command) {

case 0:

menu = 0;

break;

case 1:

writeMenu();

break;

case 2: {

unique\_ptr<Rectangle<figure\_type>> rec = make\_unique<Rectangle<figure\_type>>();

try {

cin >> rec;

st.Push(move(rec));

}

catch (const char\* exception) {

cerr << "ERROR: " << exception << '\n';

}

}

break;

case 3: {

unsigned int id;

cout << "Введите индекс элемента, который нужно удалить: ";

cin >> id;

cout << '\n';

try {

st.Remove(id);

}

catch (const char\* exception) {

cerr << "ERROR: " << exception << '\n';

}

}

break;

case 4:

for\_each(st.begin(), st.end(), print<figure\_type>);

break;

case 5: {

int max = 0;

cout << "Введите площадь: ";

cin >> max;

cout << '\n';

size\_t a = count\_if(st.begin(), st.end(), [max](unique\_ptr<Rectangle<figure\_type>>& elem) {return elem->Area() < max; });

cout << a << '\n';

}

break;

}

}

return 0;

}

**ЛИТЕРАТУРА**

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

URL: https://en.cppreference.com/w/cpp/memory/unique\_ptr (дата обращения: 16.11.2021)