**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: Айрапетова Евгения Ашотовна

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

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);

Вариант 2: квадрат на стеке

1. Описание программы

Выводится список команд, в числе которых добавление элемента в стек, удаление, вывод фигур и вывод на экран количество объектов, площадь которых меньше 100.

1. Набор тестов

Тест 1:  
0 0  
0 25  
25 25  
25 0  
  
30 30  
110 30  
110 110  
30 110  
  
Тест 2:  
45 10   
50 10   
50 15   
45 15  
  
12 3  
22 3  
22 13  
12 13  
  
30 40  
130 40  
130 140  
13 140

Тест 3:  
4 0  
14 0  
14 10  
4 10  
  
30 40  
35 40  
35 45  
30 45

1. Результаты выполнения тестов

Тест 1:  
Number of elements is 2  
Area less then 100: 0

Тест 2:  
Number of elements is 3  
Area less then 100: 1

Тест 3:  
Number of elements is 2  
Area less then 100: 1  
Pop  
Number of elements is 1  
Area less then 100: 0

1. Листинг программы

square.hpp:

#pragma once

#include <iostream>

#include <cmath>

template <class T>

struct Square {

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

static const size\_t vertexes{ 4 };

vertex\_t a, b, c, d;

Square() {

a = std::make\_pair(0, 0);

b = std::make\_pair(0, 0);

c = std::make\_pair(0, 0);

d = std::make\_pair(0, 0);

}

Square(const vertex\_t& aa, const vertex\_t& bb, const vertex\_t& cc, const vertex\_t& dd) :

a(aa), b(bb), c(cc), d(dd) {

}

Square(const Square& other) {

a = other.a;

b = other.b;

c = other.c;

d = other.d;

}

Square& operator=(const Square& other) {

a = other.a;

b = other.b;

c = other.c;

d = other.d;

return \*this;

}

};

template <class T>

std::ostream& operator<<(std::ostream& os, Square<T> sq) {

os << "[" << sq.a.first << "," << sq.a.second << "]";

os << "[" << sq.b.first << "," << sq.b.second << "]";

os << "[" << sq.c.first << "," << sq.c.second << "]";

os << "[" << sq.d.first << "," << sq.d.second << "]";

return os;

}

template <class T>

std::istream& operator>>(std::istream& is, Square<T>& sq) {

is >> sq.a.first >> sq.a.second

>> sq.b.first >> sq.b.second

>> sq.c.first >> sq.c.second

>> sq.d.first >> sq.d.second;

return is;

}

template <typename T>

T S(Square<T> sq) {

if (sq.a.first != sq.b.first) return pow(sq.a.first - sq.b.first, 2);

return pow(sq.a.second - sq.b.second, 2);

}

stack.hpp:

#pragma once

template <typename T>

class Stack

{

private:

struct Node {

Node\* prevElement; // указатель на предыдущий элемент стека

T value;

Node() {

}

Node(T val) {

value = val;

prevElement = NULL;

}

};

Node\* topElement = new Node();

int count;

public:

class StackIterator {

private:

Stack& stack;

size\_t index;

friend class Stack;

public:

StackIterator(Stack& l, int i) : stack(l), index(i) {}

StackIterator& operator++() {

++index;

return \*this;

}

T operator\*() {

return stack[index];

}

T\* operator->() {

return &stack[index];

}

bool operator!=(const StackIterator& other) {

if (index != other.index) return true;

if (&stack != &(other.stack)) return true;

return false;

}

bool operator==(const StackIterator& other) {

if (index != other.index) return false;

if (&stack != &(other.stack)) return false;

return true;

}

};

Stack() {

count = 0;

}

~Stack() {}

bool empty() {

return (count == 0);

}

int size() {

return count;

}

void push(T value) {

Node\* newElement = new Node(value);

if (count == 0) {

topElement = newElement;

}

else {

newElement->prevElement = topElement;

topElement = newElement;

}

++count;

}

//возвращаем верхний элемент стека

T top() {

return topElement->value;

}

void pop() {

if (!empty()) {

Node\* temp = topElement;

topElement = topElement->prevElement;

delete temp;

--count;

}

else {

std::cout << "The stack is empty!" << std::endl;

}

}

};

main.cpp:

#include "square.hpp"

#include "stack.hpp"

#include <iostream>

int main() {

unsigned int input;

std::cout << "1. Push\n2. Pop\n3. Top element\n4. Count elements\n5. Number of shapes with an area less than 100\n6. End\n";

Stack<Square<int>> stack;

int task = 0;

while (std::cin >> input) {

if (input == 1) {

Square<int> s;

std::cin >> s;

stack.push(s);

if (S(s) < 100) { ++task; }

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

}

else if (input == 2) {

if (!stack.empty()) {

if (S(stack.top()) < 100) { --task; }

stack.pop();

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

}

else { std::cout << "Error: stack is empty" << std::endl; }

}

else if (input == 3) {

std::cout << "Top element is " << stack.top() << std::endl;

}

else if (input == 4) {

std::cout << "Number of elements is " << stack.size() << std::endl;

}

else if (input == 5) {

std::cout << "Area less then 100: " << task << std::endl;

}

else if (input == 6) {

std::cout << "Goodbye!" << std::endl;

return 0;

}

else {

std::cout << "Error of input";

}

}

}

1. Вывод

В процессе выполнения данной лабораторной работы мне стал доступен такой инструмент, как итераторы. Также я освежила в памяти свои знания по хранению данных в стеке и реализации стека на языке C++.

Список литературы:

1. “Структура данных - стеки” [Электронный ресурс]. URL: <http://cppstudio.com/post/5155/> (дата обращения: 15.11.2020)
2. “Итераторы” [Электронный ресурс]. URL: <https://metanit.com/cpp/tutorial/7.3.php> (дата обращения: 16.11.2020)