**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: Терво Александр Александрович

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Изучить основы работы с коллекциями, познакомиться с шаблономпроектирования «Итератор». Создать шаблон динамической коллекции согласно варианту задания:

1.Коллекция должна быть реализована с помощью умных указателей.Опционально использование std::unique\_ptr;

2.В качестве параметра шаблона коллекция должна принимать тип данныхфигуры;

3.Реализовать forward\_iterator по коллекции;

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

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

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

7.При выполнении недопустимых операций (например выход аз границыколлекции или удаление несуществующего элемента) необходимогенерировать исключения;

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

9.Коллекция должна содержать метод доступа: pop, push, top;

10.Реализовать программу, которая:

• позволяет вводить с клавиатуры фигуры (с типом int в качествепараметра 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 меньшезаданной (с помощью std::count\_if).

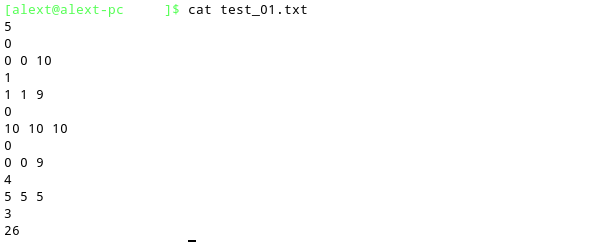
Вариант 1: треугольник, стек.

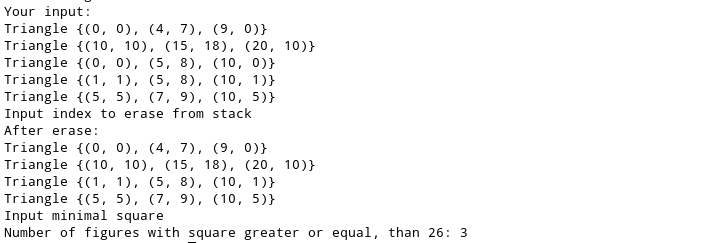
1. Описание программы

Будем использовать шаблонный класс треугольника из предыдущей лабораторной работы. Для реализации стека и итераторов используем std::shared\_ptr, чтобыпри вызове деструктора итератора не уничтожался сам стек. Для вставки и удаления из позицию итератора нужно представлять стек как список, потомучто нужно изменять указатели соседних элементов. При использовании итератора, если произошёл выход за границу или пытаемся удалить пустой итератор, будем генерировать исключение std::runtime\_error, тем самым обезопасив программу.

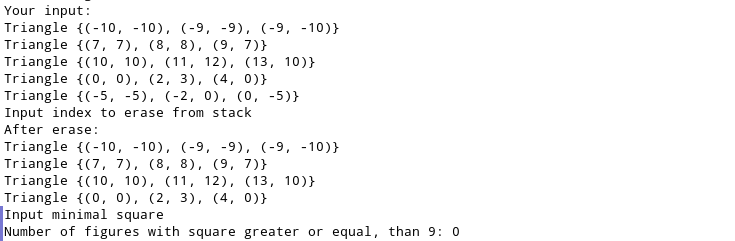
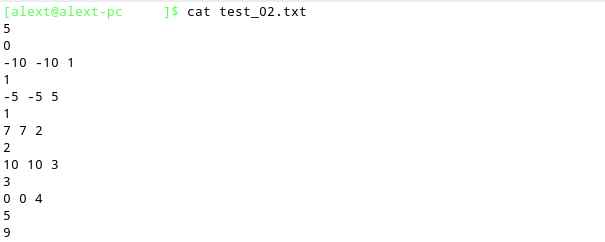
1. Набор тестов и результат их выполнения

Тест №1

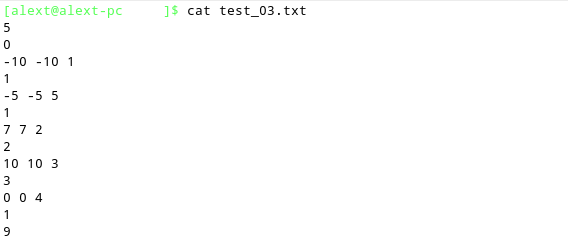


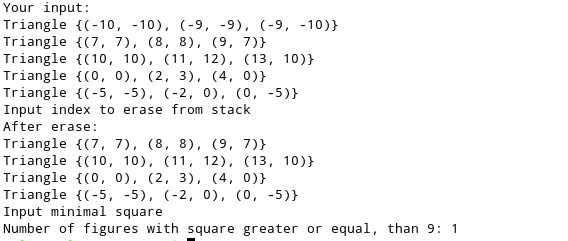


Тест №2

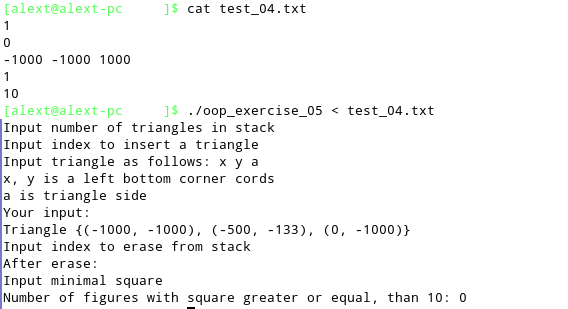


Тест №3





Тест №4



1. Листинг программы

main.cpp:

#include <iostream>

#include <vector>

#include <cmath>

#include "triangle.hpp"

#include "rectangle.hpp"

#include "quad.hpp"

int main() {

double x, y, sideX, sideY;

std::cout << "Enter triangle: \n";

std::cout << "Enter x y of vertex and side length\n";

std::cin >> x >> y >> sideX;

Triangle\* tri = new Triangle(Point(x, y), Point(x + 1 / 2 \* sideX, y + sideX \* std::pow(3, 0.5) / 2), Point(x + sideX, y));

std::vector <Figure\*> arr;

arr.push\_back(tri);

std::cout << "Enter quadrate: \n";

std::cout << "Enter x y of vertex and side length\n";

std::cin >> x >> y >> sideX;

Quad\* quad = new Quad(Point(x, y), Point(x + sideX, y), Point(x + sideX, y + sideX), Point(x, y + sideX));

arr.push\_back(quad);

std::cout << "Enter rectangle: \n";

std::cout << "Enter x y of vertex, horizontal and vertical sides\n";

std::cin >> x >> y >> sideX >> sideY;

Rectangle\* rect = new Rectangle(Point(x, y), Point(x + sideX, y), Point(x + sideX, y + sideX), Point(x, y + sideX));

arr.push\_back(rect);

std::cout << std::endl;

//3 basic functions

Point p(0.0, 0.0);

for (auto fig : arr) {

fig->printVertices();

p = fig->getCenter();

std::cout << "Geometric center is (" << p.x << ", " << p.y << ")\n";

std::cout << "Square is equal to " << fig->getSquare() << "\n\n";

}

//sum of squares

double square = 0.0;

for (auto fig : arr) {

square += fig->getSquare();

}

std::cout << "Total square is equal to " << square << "\n\n";

//deleting element by index

int index;

std::cout << "Enter index of element to delete\n";

std::cin >> index;

arr.erase(arr.begin() + index);

std::cout << "After delete:\n";

for (auto fig : arr) {

fig->printVertices();

p = fig->getCenter();

std::cout << "\nGeometric center is (" << p.x << ", " << p.y << ")\n";

std::cout << "Square is equal to " << fig->getSquare() << "\n\n";

}

square = 0.0;

for (auto fig : arr) {

square += fig->getSquare();

}

std::cout << "Total square is equal to " << square << "\n\n";

}

stack.hpp:

#ifndef STACK\_HPP

#define STACK\_HPP

#include <iostream>

#include <memory>

template<class T>

class TStackNode;

template<class T>

class TStack;

template<class T>

void operator ++ (std::shared\_ptr< TStackNode<T> > & curStackNode) {

if (curStackNode) {

curStackNode = curStackNode->Next;

} else {

throw(std::runtime\_error("Iterator points to nullptr!"));

}

}

template<class T>

bool operator != (const TStackNode<T> & lhs, const TStackNode<T> & rhs) {

return &lhs.Data != &rhs.Data;

}

template<class T>

bool operator == (const TStackNode<T> & lhs, const TStackNode<T> & rhs) {

return &lhs.Data == &rhs.Data;

}

template<class T>

std::ostream & operator << (std::ostream & out, const TStackNode<T> & curStackNode) {

out << curStackNode.Data;

return out;

}

template<class T>

class TStackNode {

public:

T Data;

std::shared\_ptr< TStackNode<T> > Next;

public:

TStackNode() noexcept : Data(), Next(nullptr) {};

explicit TStackNode(const T & elem) noexcept : Data(elem), Next(nullptr) {};

friend void operator ++ <> (std::shared\_ptr< TStackNode<T> > & curStackNode);

friend bool operator != <> (const TStackNode<T> & lhs, const TStackNode<T> & rhs);

friend bool operator == <> (const TStackNode<T> & lhs, const TStackNode<T> & rhs);

friend std::ostream & operator << <> (std::ostream & out, const TStackNode<T> & curStackNode);

friend class TStack<T>;

};

template<class T>

class TStack {

private:

std::shared\_ptr< TStackNode<T> > TopNode;

public:

class iterator {

private:

std::shared\_ptr< TStackNode<T> > ptr;

public:

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

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

using value\_type = T;

using pointer = T\*;

using reference = T&;

iterator() : ptr(nullptr) {}

iterator(const std::shared\_ptr< TStackNode<T> > & anotherPtr) : ptr(anotherPtr) {}

bool IsNullptr() {

return ptr == nullptr;

}

friend void operator ++ (iterator & it) {

++it.ptr;

}

friend bool operator != (const iterator & lhs, const iterator & rhs) {

return lhs.ptr != rhs.ptr;

}

friend std::ostream & operator << (std::ostream & out, const iterator & it) {

out << \*it.ptr;

return out;

}

TStackNode<T>& operator \* () {

return \*ptr;

}

};

iterator begin();

iterator end();

TStack() noexcept : TopNode(nullptr) {};

void Pop();

void Push(const T & elem);

T Top();

void Erase(iterator it);

void Insert(iterator it, const T & elem);

};

template<class T>

typename TStack<T>::iterator TStack<T>::begin() {

return TStack<T>::iterator(TopNode);

}

template<class T>

typename TStack<T>::iterator TStack<T>::end() {

return TStack<T>::iterator(nullptr);

}

template<class T>

void TStack<T>::Pop() {

if (TopNode) {

TopNode = TopNode->Next;

} else {

throw(std::runtime\_error("Stack is empty!"));

}

}

template<class T>

void TStack<T>::Push(const T & elem) {

TStackNode<T>\* newNode = new TStackNode(elem);

std::shared\_ptr< TStackNode<T> > newNodeShared(newNode);

newNodeShared->Next = TopNode;

TopNode = newNodeShared;

}

template<class T>

T TStack<T>::Top() {

if (TopNode) {

return TopNode->Data;

} else {

throw(std::runtime\_error("Stack is empty!"));

}

}

template<class T>

void TStack<T>::Erase(TStack<T>::iterator it) {

if (it.IsNullptr()) {

throw(std::runtime\_error("Iterator points to nullptr!"));

} else {

if (\*it == \*TopNode) {

TopNode = TopNode->Next;

} else {

std::shared\_ptr< TStackNode<T> > prevNode = TopNode;

while (\*prevNode->Next != \*it) {

++prevNode;

}

prevNode->Next = prevNode->Next->Next;

}

}

}

template<class T>

void TStack<T>::Insert(TStack<T>::iterator it, const T & elem) {

TStackNode<T>\* newNode = new TStackNode(elem);

std::shared\_ptr< TStackNode<T> > newNodeShared(newNode);

if (TopNode) {

if (\*it == \*TopNode) {

newNodeShared->Next = TopNode;

TopNode = newNodeShared;

return;

}

std::shared\_ptr< TStackNode<T> > prevNode = TopNode;

if (it.IsNullptr()) {

while (prevNode->Next != nullptr) {

prevNode = prevNode->Next;

}

prevNode->Next = newNodeShared;

} else {

while (\*prevNode != \*it) {

prevNode = prevNode->Next;

}

newNodeShared->Next = prevNode->Next;

prevNode->Next = newNodeShared;

std::swap(prevNode->Data, prevNode->Next->Data);

}

} else {

TopNode = newNodeShared;

}

}

#endif

triangle.hpp:

#ifndef TRIANGLE\_HPP

#define TRIANGLE\_HPP

#include <iostream>

#include <tuple>

#include <cmath>

template<class T>

struct TTriangle {

/\* Cords of left bottom corner, side \*/

std::pair<T, T> Cord;

T Side;

TTriangle(const std::pair<T, T> & cord, T side) : Cord(cord), Side(side) {}

};

template<class T>

T CalcSquare(const TTriangle<T> & Tri) {

return Tri.Side \* Tri.Side \* pow(3, 0.5) / 4;

}

template<class T>

std::ostream & operator << (std::ostream & out, const TTriangle<T> & tri) {

out << "Triangle {";

out << std::pair<T, T>(tri.Cord.first, tri.Cord.second) << ", ";

out << std::pair<T, T>(tri.Cord.first + 0.5 \* tri.Side, tri.Cord.second + tri.Side \* (std::pow(3, 0.5) / 2)) << ", ";

out << std::pair<T, T>(tri.Cord.first + tri.Side, tri.Cord.second);

out << "}";

return out;

}

template<class T1, class T2>

std::ostream & operator << (std::ostream & out, const std::pair<T1, T2> & p) {

out << "(" << p.first << ", " << p.second << ")";

return out;

}

#endif

5. Выводы

В ходе выполнения работы я узнал, как реализовать итераторы на языке C++, чтобы они были совместимы со стандартными функциями. Узнал про умные указатели std::unique\_ptr, std::shared\_ptr и в чём их разница. Изучил работу std::for\_each и std::count\_if и реализовал их взаимодействие с моей коллекцией.

6. Список литературы

1.std::shared\_ptr – cppreference.com

URL: <https://en.cppreference.com/w/cpp/memory/shared_ptr>

2.Делаем свой итератор / Хабр — Habr.com

URL: <https://habr.com/ru/post/265491/>

3.std::for\_each – cppreference.com

URL: https://en.cppreference.com/w/cpp/algorithm/for\_each

4.std::count, std::count\_if – cppreference.com

URL: https://en.cppreference.com/w/cpp/algorithm/count