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: Суханов Егор Алексеевич
Группа: 80-206
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. Так как мой вариант задания 25, мне нужно сделать класс треугольника, который:

1. Имеет параметр шаблона: скалярный тип осей системы координат;
2. Имеет перегруженные операторы ввода-вывода;
3. Имеет метод вычисления площади.

Создать шаблон динамической коллекции, 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 должна 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 с помощью std::for_each;
    - Выводит на экран количество объектов, у которых площадь меньше заданной

(с помощью std::count_if);

Так как мой вариант задания 25, мне нужно реализовать динамический массив (вектор), который:

1. Использует умные указатели;
2. Имеет шаблонный тип элемента;
3. Использует стандартные итераторы;
4. Метод вставки insert, метод удаления erase;
5. Использование исключений;
6. Доступ к элементу по оператору [].

Опишем порядок выполнения работы:

1. Изучение теоретической информации:
   - Умные указатели;
   - Итераторы в C++;
   - Стандартные алгоритмы работы с коллекциями;
2. Реализация шаблона фигуры;
3. Реализация шаблона вектора;
4. Интерактивный ввод-вывод;

2. Описание программы

Ссылка на GITHUB: https://github.com/Reterer/oop_exercise_05

Программа состоит из 3-ех компонентов:

◦ Интерактивный ввод-вывод. Находится в файле main.cpp. В цикле происходит считывание команды и последующее ее выполнение. Для каждой команды реализована своя функция. Алгоритм работы данного компонента не отличается от работы аналогичных компонентов в прошлых лабораторных работах;

◦ Класс Triangle. Данный шаблонный класс имплементирует работу с треугольниками. Он позволяет узнать координаты вершин, площадь, а также обеспечивает ввод-вывод. Треугольник строится по двум точкам: вершине и центру описанной окружности. Этот класс использует вспомогательные шаблонные функции для работы с точками. Все функции и классы для работы с треугольниками находятся в файле figure.hpp;

◦ Динамический массив. Находится в файле vector.hpp. Данный класс имеет перегруженный оператор квадратных скобок, для обращения к i ому элементу; Функцию для вставки в конец; И для получения размера массива. Кроме этого, данный класс поддерживает forward iterator, что позволяет использовать стандартные алгоритмы для работы с ним. Реализация vector представляет собой типичную

реализацию динамического массива (с буфером, размером и размером занятой части и общей ёмкостью). Данный класс был разработан с помощью подхода TDD. Для тестирования я использовал библиотеку google test.

Координаты вводятся как два последовательных числа. Для ввода треугольника нужно ввести сначала центр описанной окружности, затем одну из его вершин. Для получения справки по командам нужно ввести help.

3. Набор и результаты выполнения тестов

Unit-тесты находятся в папке tests. Тесты для интерактивного ввода-вывода находятся в папке general_tests.

test_01.txt -- проверка работы команд: вставки, удаления, вывода, вычисления кол-ва подходящих фигур.

```
insert 0 0 0 1 1
insert 1 0 0 2 2
insert 0 0 0 3 3
insert 1 0 0 3 3
print
square 2
square 20
square 50
erase 3
erase 1
erase 0
erase 0
print
```

Результаты выполнения:

```
Координаты треугольника: (3 , 3) (-4.09808 , 1.09808) (1.09808 , -4.09808)
площадь: 23.3827
Координаты треугольника: (3 , 3) (-4.09808 , 1.09808) (1.09808 , -4.09808)
площадь: 23.3827
Координаты треугольника: (1 , 1) (-1.36603 , 0.366025) (0.366025 , -1.36603)
площадь: 2.59808
Координаты треугольника: (2 , 2) (-2.73205 , 0.732051) (0.732051 , -2.73205)
площадь: 10.3923
Кол-во фигур, площадь которых меньше 2, равно: 0
Кол-во фигур, площадь которых меньше 20, равно: 2
Кол-во фигур, площадь которых меньше 50, равно: 4
```

test_02.txt -- проверка обработки ошибок.

```
sjehfkj kjefh w
insert 0 0 0 0 0
insert 0 kehfkewjfh 0 0 0 0
insert 0 kehfkewjfh 0 0 1 1
insert 1 0 0 1 1
erase 0
square kefjh
```

Результаты выполнения:

```
Введена неизвестная команда. Чтобы вывести справку, введите "help"
Введены некорректные координаты.
Введены некорректные координаты.
Введены некорректные координаты.
Введено некорректное число.
Введено некорректное число.
Введено некорректное число.
```

4. Листинг программы

файл **main.cpp**:

```
/*
  Лабораторная работа: 5
  Вариант: 21
  Группа: М8О-206Б-19
  Автор: Суханов Егор Алексеевич

  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.

  Фигура:
       Треугольник
  Структура данных:
       Динамический массив

*/
#include <iostream>
#include <algorithm>
#include "vector.hpp"
```

```
#include "figure.hpp"


void clear()
{
	std::cin.clear();
	std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}

void help()
{
	std::cout <<
		"команды:\n"
		"    help        -- выводит этот текст\n"
		"    exit        -- завершает работу программы\n"
		"    print       -- выводит вектор фигур (выполняет их методы)\n"
		"    square <par>  -- выводит кол-во элементов, площадь которых меньше
или равна заданному параметра\n"
		"    erase <id> -- удаляет элемент с индексом id\n"
		"    insert <id> <center> <vertex> -- вставляет треугольник на позицию
с номером id\n"
		<< std::endl;
}

template<class T>
void print(Vector<T>& figures) {
	std::for_each(figures.begin(), figures.end(), [](T& val) {
		std::cout << val;
	});
}

template<class T>
void square(Vector<T>& figures) {
	double par;
	if (!(std::cin >> par)) {
		std::cout << "Введено некоректное число.\n";
		clear();
		return;
	}

	long long count = std::count_if(figures.begin(), figures.end(), [par](T&
val){
		return val.Square() <= par;
	});
	std::cout << "Кол-во фигур, площадь которых меньше " << par << ", равно: "
<< count << '\n';
}

template<class T>
void erase(Vector<T>& vec) {
	int del_element_idx;
	if (!(std::cin >> del_element_idx) || del_element_idx < 0 || del_element_idx
>= vec.size()) {
		std::cout << "Введено некоректное число.\n";
```

```
			clear();
			return;
		}

		vec.erase(vec.iterator_by_index(del_element_idx));
}

template<class T>
void insert(Vector<T>& vec) {
		int new_element_index;
		if (!(std::cin >> new_element_index) || new_element_index < 0 ||
new_element_index > vec.size()) {
			std::cout << "Введено некоректное число.\n";
			clear();
			return;
		}

		T new_el;
		if (!(std::cin >> new_el))
		{
			std::cout << "Введена некоректные координаты.\n";
			clear();
			return;
		}

		vec.insert(vec.iterator_by_index(new_element_index), new_el);
}

int main()
{
		setlocale(LC_ALL, "russian");
		Vector<Triangle<float> > figures;

		std::string cmd;
		std::cout << '>';
		while (std::cin >> cmd)
		{
			if (cmd == "help")
				help();
			else if (cmd == "exit")
				break;
			else if (cmd == "print")
				print(figures);
			else if (cmd == "square")
				square(figures);
			else if (cmd == "erase")
				erase(figures);
			else if (cmd == "insert")
				insert(figures);
			else
			{
				std::cout << "Введена неизвестная команда. Чтобы вывести
справку, введите \"help\"" << std::endl;
				clear();
```

```
            }

            std::cout << '>';
      }

      return 0;
}
```

файл **figure.hpp**:

```
#pragma once
#include <stdexcept>
#include <utility>
#include <iostream>

const double PI = 3.141592653589793;
// Векторное сложение
template <class T>
std::pair<T, T> operator + (const std::pair<T, T> a, const std::pair<T, T> b)
{
      return { a.first + b.first, a.second + b.second };
}
// Векторное вычитание
template <class T>
std::pair<T, T> operator - (const std::pair<T, T> a, const std::pair<T, T> b)
{
      return { a.first - b.first, a.second - b.second };
}
// Скалярное умножение векторов
template <class T>
T operator * (const std::pair<T, T> a, const std::pair<T, T> b) {
      return a.first * b.first + a.second * b.second;
}
// Умножение на скаляр
template <class T>
std::pair<T, T> operator * (const T a, const std::pair<T, T> b) {
      return { a * b.first, a * b.second };
}

// Вывод координат точки
template <class T>
std::ostream& operator<< (std::ostream& out, const std::pair<T, T>& p) {
      out << '(' << p.first << " , " << p.second << ')';
      return out;
};

// Ввод координат
template <class T>
std::istream& operator>> (std::istream& in, std::pair<T, T>& p) {
      in >> p.first >> p.second;
```

```
	return in;
}

// Поворот вектора pair на угол angle
template <class T>
std::pair<T, T> rotate(std::pair<T, T> pair, const double angle) {
	std::pair<T, T> rotated;
	rotated.first = (T)(pair.first * std::cos(angle) - pair.second *
std::sin(angle));
	rotated.second = (T)(pair.first * std::sin(angle) + pair.second *
std::cos(angle));
	return rotated;
}


template<typename T>
class Triangle {
public:
	using Vertex = std::pair<T, T>;

	Triangle();
	Triangle(Vertex center, Vertex vertex);

	// Возвращает площадь треугольника
	double Square() const;

	friend std::ostream& operator<< (std::ostream& out, const Triangle<T>&
t) {
		const double ANGLE = 2 * PI / Triangle<T>::vertex_count; // угол
между двумя соседними вершинами и центром

		out << "Координаты треугольника: ";
		out << t.vertex;		// Выводим первую вершину
		auto vec = t.vertex - t.center;
		for (int i = 2; i <= Triangle<T>::vertex_count; ++i)
		{
			vec = rotate(vec, ANGLE); // Получаем координаты следующий
вершины
			out << ' ' << t.center + vec;
		}
		out << "\t площадь: " << t.Square() << '\n';
		return out;
	}

	friend std::istream& operator>> (std::istream& in, Triangle<T>& t) {
		in >> t.center >> t.vertex;
		if (t.center == t.vertex)
			in.setstate(std::ios_base::failbit);
		return in;
	}
```

```
private:
	static const int vertex_count = 3; // Кол-во вершин треугольника

	Vertex center; // Центр описанной окружности
	Vertex vertex; // Одна из вершин треугольника
};

template<typename T>
Triangle<T>::Triangle()
{}

template<typename T>
Triangle<T>::Triangle(Vertex center, Vertex vertex)
	: center{ center }, vertex{ vertex }
{
	if (center == vertex)
		throw std::invalid_argument("center cannot be eq to the
vertex");
}

template<typename T>
double Triangle<T>::Square() const {
	auto vecRadius = vertex - center; // Вектор-радиус описанной
окружности
	double sqRadius = vecRadius * vecRadius;	// Квадрат радуиса
описанной окружности
	return vertex_count / 2. * sqRadius * std::sin(2 * PI / vertex_count);
}
```

Файл **vector.hpp**:

```
#pragma once
#include <stdexcept>
#include <memory>

template<typename T>
class Vector {
public:
	explicit Vector()
		: buf_{ nullptr }, size_{ 0 }, cap_{ 0 }
	{}
	explicit Vector(const int size)
		: Vector(size, size)
	{}
	explicit Vector(const int size, const int cap) {
		this->size_ = size;
		this->cap_ = cap;
		if (size_ > cap_)
			throw std::invalid_argument("size_ of vector can't be more
then capacity.");
		if (cap_ == 0)
```

```
            this->buf_ = std::unique_ptr<T>{ nullptr };
      else
            this->buf_ = std::unique_ptr<T>{new T[cap_]};
}

Vector(const std::initializer_list<T>& list)
      : Vector(list.size())
{
      int count = 0;
      for (const T& el : list) {
            this->buf_.get()[count] = el;
            ++count;
      }
}

Vector(const Vector<T>& vec)
      : Vector(vec.size_, vec.cap_)
{
      this->buf_ = std::unique_ptr<T>{ new T[cap_] };
      for (int i = 0; i < vec.size_; i++) {
            this->buf_[i] = vec[i];
      }
}

int size() const {
      return this->size_;
}

T& operator[] (const int i) {
      if (i < 0 || i >= this->size_)
            throw std::out_of_range("Out of range");
      return this->buf_.get()[i];
}

// Итераторы
class iterator
{
      friend class Vector;
public:
      using iterator_category = std::forward_iterator_tag;
      using value_type = T;
      using difference_type = int;
      using pointer = T*;
      using reference = T&;

      bool operator== (const iterator& it) const {
            return it.vec == vec && it.ptr == ptr;
      }
      bool operator!= (const iterator& it) const {
            return it.vec != vec || it.ptr != ptr;
      }
```

```
        T& operator* () {
                is_valid();
                return *ptr;
        }
        T* operator-> () {
                is_valid();
                return ptr;
        }

        iterator& operator++() {
                is_valid();
                ptr++;
                return *this;
        }
        iterator operator++(int) const {
                is_valid();
                return { vec, ptr + 1 };
        }

private:
        void is_valid() const {
                if (vec == nullptr || ptr == nullptr)
                        throw std::runtime_error("Nullptr iterator");
                if (vec->buf_.get() + vec->size_ <= ptr)
                        throw std::out_of_range("Iterator gt or eq end");
        }

        Vector<T>* vec;
        T* ptr;
};
iterator begin() {
        iterator it;
        it.vec = this;
        it.ptr = buf_.get();
        return it;
}
iterator end() {
        iterator it;
        it.vec = this;
        it.ptr = buf_.get() + size_;
        return it;
}
// Возвращает итератор, который указыавет на элемент под номером index
iterator iterator_by_index(int index) {
        if (index < 0 || index > size_)
                throw std::out_of_range("bad index");

        iterator it;
        it.vec = this;
        it.ptr = buf_.get() + index;
        return it;
```

```
	}
	void push_back(const T& val) {
		if (size_ >= cap_)
			grow();
		buf_.get()[size_] = val;
		++size_;
	}
	void push_back(T&& val) {
		if (size_ >= cap_)
			grow();
		buf_.get()[size_] = val;
		++size_;
	}
	iterator insert(const iterator& position, const T& value) {
		if (position.vec != this || buf_.get() + size_ < position.ptr)
			throw std::invalid_argument("Invalid iterator");

		if (position == end()) {
			push_back(value);
			return end();
		}

		int new_element_idx = (int)(position.ptr - buf_.get());
		T temp = buf_.get()[new_element_idx];
		buf_.get()[new_element_idx] = value;
		for(int i = new_element_idx + 1; i < size_; i++) {
			std::swap(temp, buf_.get()[i]);
		}
		push_back(temp);

		return iterator_by_index(new_element_idx);
	}
	iterator erase(const iterator& position) {
		if (position.vec != this || buf_.get() + size_ <= position.ptr)
			throw std::invalid_argument("Invalid iterator");

		int del_element_idx = (int)(position.ptr - buf_.get());
		T temp = buf_.get()[size_ - 1];
		for (int i = size_ - 2; i >= del_element_idx; i--) {
			std::swap(temp, buf_.get()[i]);
		}

		--size_;
		return iterator_by_index(del_element_idx);
	}

private:
	void grow() {
		int new_cap = (cap_ == 0) ? 1 : cap_ * 2;
		T* new_buf = new T[new_cap];
		for (int i = 0; i < size_; i++) {
```

```
                new_buf[i] = buf_.get()[i];
            }

            buf_ = std::unique_ptr<T>(new_buf);
            cap_ = new_cap;
        }

        std::unique_ptr<T> buf_;
        int size_;
        int cap_;
};
```

5. Выводы

В данной лабораторной работе я научился использовать умные указатели, а также тестировать программы в visual studio. Умные указатели сильно упрощают работу с “сырыми” указателями. Они используют идиому RAII, которая гласит: “Захват ресурса -- есть инициализация”. Другими словами память будет выделяться в конструкторе указателя, а в его деструкторе -- освобождаться. Это избавляет программиста от головной боли: “не забыть освободить память перед return” или “как же освободить память в случае возникновения исключения”. Итераторы позволяют использовать стандартные алгоритмы. С помощью них очень удобно перебирать все элементы коллекции.

6. Список литературы

1. Страуструп, Бьёрн. Язык программирования C++. Краткий курс, 2-е изд. : Пер. с англ. - СПб.: ООО "Диалектика", 2019. - 320 с.: ил. - Парал. тит. англ.

2. Стандартная библиотека, Итераторы[Электронный ресурс]. URL: https://ru.cppreference.com/w/cpp/iterator (дата обращения 27.11.20).