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: Будникова Валерия Павловна
Группа: 80-207
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2020

1. Постановка задачи

ВАРИАНТ 31

Создать шаблон динамической коллекцию, согласно варианту задания:

1. Коллекция должна быть реализована с помощью умных указателей.
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.
6. Коллекция должна содержать метод удаления из позиции итератора.
7. При выполнении недопустимых операций необходимо генерировать исключения.
8. Итератор должен быть совместим со стандартными алгоритмами.
9. Коллекция должна содержать метод доступа: доступ к элементу по оператору [].
10. Реализовать программу, которая:
    a. позволяет вводить с клавиатуры фигуры и добавлять в коллекцию;
    b. позволяет удалять элемент из коллекции по номеру элемента;
    c. выводит на экран введенные фигуры с помощью std::for_each;
    d. выводит на экран количество объектов, у которых площадь меньше заданной (с помощью std::count_if).

Фигура по варианту: 6-угольник.

Контейнер по варианту: Динамический массив.

2. Описание программы

Для хранения фигуры создается шаблонный класс Hexagon, который хранит в себе координату верхней точки фигуры и длину стороны фигуры. Для хранения коллекции используется умный указатель. При инициализации класса Vect(Динамический массив), выделяется память под элементы типа Hexagon. Отдельно реализован шаблонный класс iterator, в нем также используются умные указатели. Для того, чтобы итератор был совместим со стандартными алгоритмами (которые мы используем в данной программе: std::for_each, std::count_if), необходимо перегрузить операторы - ==, !=, ++, *, +. В классе Vector реализованы методы begin() и end(), которые возвращают итератор. Также метод insert - получает на вход итератор и элемент, который нужно вставить - вставляет элемент на нужную позицию. Реализован метод erase - получает на вход итератор - удаляет элемент с нужной позиции. Фигуры печатаются на экран с помощью std::for_each. Элементы добавляются и удаляются из коллекции с помощью итераторов(методов insert и erase). Также при выполнении недопустимых операций генерируются исключения.

3. Руководство по использованию программы

Взаимодействие с пользователем происходит с помощью меню:

Введите:
1 - Чтобы добавить фигуру
2 - Чтобы распечатать все фигуры
3 - Вставить по номеру
4 - Удалить по номеру
5 - Количество фигур, у которых площадь меньше заданной
6 - Завершить

Отсчет элементов начинается с нулевой позиции. При вставке по номеру, элемент будет вставляться после номера, который введет пользователь.

4. Набор тестов и результаты работы программы

Описание ввода: Пользователь сначала добавляет фигуры в коллекцию, печатая коллекцию на экран после каждого добавления. Затем Вставляет по номеру и удаляет по номеру фигуры из коллекции, на каждом шаге печатая на экран коллекцию. В конце пользователь запрашивает у программы число фигур, площадь которых меньше заданной(вводит значение площади).

```
Введите:
 1 - Чтобы добавить фигуру
 2 - Чтобы распечатать все фигуры
 3 - Вставить по номеру
 4 - Удалить по номеру
 5 - Количество фигур, у которых площадь меньше заданной
 6 - Завершить
Введите команду:1
Введите координаты верхней точки и длину стороны через пробел: 1 2
3

Введите команду:2
(1,2)(-1,0)(3,0)
(-1,-2)(3,-2)(1,-4)

Введите команду:1
Введите координаты верхней точки и длину стороны через пробел: 3
4 7

Введите команду:2
(1,2)(-1,0)(3,0)
(-1,-2)(3,-2)(1,-4)

(3,4)(-3,0)(9,0)
(-3,-6)(9,-6)(3,-10)
```

```
Введите команду:3
Введите номер позиции:2
Введите координаты верхней точки и длину стороны через пробел: 2 2
9

Введите команду:2
(1,2)(-1,0)(3,0)
(-1,-2)(3,-2)(1,-4)

(3,4)(-3,0)(9,0)
(-3,-6)(9,-6)(3,-10)

(2,2)(-5,-2)(9,-2)
(-5,-11)(9,-11)(2,-16)

Введите команду:4
Введите номер позиции:1

Введите команду:2
(1,2)(-1,0)(3,0)
(-1,-2)(3,-2)(1,-4)

(2,2)(-5,-2)(9,-2)
(-5,-11)(9,-11)(2,-16)

Введите команду:5
Введите площадь:100
Кол-во фигур: 1
Введите команду:5
Введите площадь:500
Кол-во фигур: 2
Введите команду:6
```

6. Листинг программы

main.cpp

```
#include <cmath>
#include <algorithm>
#include <iostream>
#include "iterator.hpp"
#include "vector.hpp"

void Menu(){
     std::cout << "Введите:\n 1 - Чтобы добавить фигуру\n 2 - Чтобы распечатать все фигуры\n 3 - Вставить по номеру\n";
     std::cout << " 4 - Удалить по номеру\n 5 - Количество фигур, у которых площадь меньше заданной\n 6 - Завершить\n";
}

void Print(){
    std::cout << "Введите координаты верхней точки и длину стороны через пробел: ";
}
```

```
int main() {
    Vect<int> Vector;
    auto print = [](const Hexagon<int>& ob){
        std::cout << ob << std::endl << std::endl;
    };
    Menu();
    int x1 = 0, y1 = 0, side = 0;
    int m;
    std::cout << "Введите команду:";
    while (std::cin >> m && m < 6) {
        switch (m) {
            case 1: {
                Print();
                std::cin >> x1 >> y1 >> side;
                if (side < 0) {
                    std::cout << "Введенные значения не верные" << std::endl;
                    break;
                }
                Vector.push_back(Hexagon(x1, y1, side));
                break;
            }
            case 2: {
                std::for_each(Vector.begin(), Vector.end(), print);
                break;
            }
            case 3: {
                int pos;
                std::cout << "Введите номер позиции:";
                std::cin >> pos;
                try {
                    if (pos < 0) {
                            throw std::out_of_range("Iterator cannot be incremented
past the end of range.");
                    } else {
                        Print();
                        std::cin >> x1 >> y1 >> side;
                        Vector.insert(Vector.begin() + pos, Hexagon(x1, y1, side));
                    }
                } catch (std::out_of_range &err) {
                    std::cout << err.what();
                }
                break;
            }
            case 4: {
                int pos;
                std::cout << "Введите номер позиции:";
                std::cin >> pos;
                try {
                    Vector.erase(Vector.begin() + pos);
                } catch (std::out_of_range &error) {
                    std::cout << error.what();
                }
                break;
            }
```

```
            case 5: {
                int S;
                std::cout << "Введите площадь:";
                std::cin >> S;
                auto square = [S](const Hexagon<int>& ob){
                    return (3 * sqrt(3) * ob.side * ob.side / 2) < S;
                };
                    std::cout << "Кол-во фигур: " << std::count_if(Vector.begin(),
Vector.end(), square);
                break;
            }
            default:
                break;
        }
        std::cout << "\nВведите команду:";
    }
    return 0;
}
```

iterator.hpp

```
#include <memory>
#include <iostream>

template<class T>
class iterator{
    public:
        using self_type = iterator<T>;
        using iterator_category = std::forward_iterator_tag;
        using difference_type = std::ptrdiff_t;
        using value_type = T ;
        using pointer = T*;
        using reference = T&;
    private:
        std::shared_ptr<T> ptr;
        size_t ind;
        size_t Size;
        bool compatible(self_type const &other) const {
            return ptr == other.ptr;
        }
    public:
        iterator(): ind(0), Size(0) {}
        iterator(std::shared_ptr<T> fig, size_t ind1, size_t s): ptr(fig),
ind(ind1), Size(s) {}
        ~iterator(){}
        bool operator==(self_type const &other) const {
            assert(compatible(other));
            return ind == other.ind;
        }

        bool operator!=(self_type const &other) const{
            return !(*this == other);
        }

        self_type &operator++(){
            if (ind >= Size) {
```

```
                throw std::out_of_range("Iterator cannot be incremented past the
end of range.");
                ind = Size;
                return *this;
            } else {
                ++ind;
                return *this;
            }
        }

        self_type operator++(int){
            self_type tmp = *this;
            ++*this;
            return tmp;
        }

        reference operator*() const{
            if (ptr == nullptr) {
                throw std::bad_function_call();
            } else {
                return ptr.get()[ind];
            }
        }

        self_type operator+(difference_type offset) {
            if (ind + offset > Size) {
                throw std::out_of_range("Iterator cannot be incremented past the
end of range.");
            } else {
                ind += offset;
            }
            return *this;
        }
};
```

vector.hpp

```
#include <memory>
#include <iostream>
template<class T>
struct Hexagon {
    std::pair<T, T> point;
    Hexagon(T x, T y, T side1): point(x, y), side(side1) {}
    Hexagon(): point(0,0), side(0) {}
    T side;
};
template<class T>
std::ostream& operator<<(std::ostream &os, std::pair<T,T> p) {
    os << '(' << p.first << ',' << p.second << ')';
    return os;
}
template<class T>
std::ostream& operator<<(std::ostream &os, Hexagon<T> p) {
    std::pair<T, T> p1 = p.point;
    std::pair<T, T> p2(p.point.first - (double)p.side * sqrt(3) / 2, p.point.second
- (double) p.side / 2);
    std::pair<T, T> p3(p.point.first + (double)p.side * sqrt(3) / 2, p.point.second
- (double) p.side / 2);
    std::pair<T, T> p4(p.point.first - (double)p.side * sqrt(3) / 2, p.point.second
- (double) p.side / 2 - p.side);
    std::pair<T, T> p5(p.point.first + (double)p.side * sqrt(3) / 2, p.point.second
- (double) p.side / 2 - p.side);
```

```
    std::pair<T, T> p6(p.point.first, p.point.second - 2 * p.side);
    os << p1 << p2 << p3 << std::endl << p4 << p5 << p6;
    return os;
}
template<class T>
class Vect {
    public:
        using Hex = Hexagon<T>;
        Vect(){
            size = 0;
            capacity = 2;
        };
        ~Vect(){};
        void push_back(const Hex fig){
            if (size == capacity) {
                capacity <<= 1;
                std::shared_ptr<Hex> ptr1{new Hex[capacity]};
                for (int i = 0; i < size; ++i) {
                    ptr1.get()[i] = ptr.get()[i];
                }
                ptr1.get()[size] = fig;
                ++size;
                ptr = ptr1;
            } else {
                ptr.get()[size] = fig;
                ++size;
            }
        }
        Hex operator[](int i) {
            return ptr.get()[i];
        }
        iterator<Hex> begin() {
            return iterator(ptr, 0, size);
        }
        iterator<Hex> end() {
            return iterator(ptr, size, size);
        }

        void insert(iterator<Hex> iter, Hex fig) {
            iterator<Hex> iter_tmp = iterator(ptr, 0, size);
            std::shared_ptr<Hex> ptr1{new Hex[capacity + 1]};
            int i = 0;
            while(iter != iter_tmp) {
                ++iter_tmp;
                ptr1.get()[i] = ptr.get()[i];
                ++i;
            }
            ptr1.get()[i] = fig;
            ++i;
            ++size;
            while(i < size) {
                ptr1.get()[i] = ptr.get()[i - 1];
                ++i;
            }
            ptr = ptr1;
        }
        void erase(iterator<Hex> iter) {
            iterator<Hex> iter_tmp = iterator(ptr, 0, size);
            std::shared_ptr<Hex> ptr1{new Hex[capacity]};
            int i = 0;
            while(iter != iter_tmp) {
                ++iter_tmp;
                ptr1.get()[i] = ptr.get()[i];
                ++i;
            }
            ++i;
```

```
            while(i < size) {
                ptr1.get()[i - 1] = ptr.get()[i];
                ++i;
            }
            --size;
            ptr = ptr1;
        }

    private:
        std::shared_ptr<Hex> ptr{new Hex[2]};
        int size;
        int capacity;
};
```

makefile

```
CC=g++

CFLAGS=-std=c++17

OUTPUT=lab5


all:

	$(CC) $(CFLAGS) main.cpp -o $(OUTPUT)

run:

	./$(OUTPUT)
```

7. Выводы

В данной лабораторной работе я научилась реализовывать итератор для динамического массива, изучила и проверила на практике его совместимость со стандартными алгоритмами, которые использовались в программе. Также я научилась работать с умными указателями и попрактиковалась с созданием и использованием лямбда-выражений.

Список литературы

1. Презентация — “Умные указатели ЛЕКЦИЯ №8” Д. В. Дзюба (дата обращения: 14.11.20)
2. cppreference.com [Электронный ресурс]. URL: https://en.cppreference.com/w/ (дата обращения: 14.11.20)