Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа
по курсу «Объектно-ориентированное программирование»
III Семестр

Задание 5
Вариант 19
Основы работы с коллекциями: итераторы

Студент:	Овечкин В.А.
Группа:	М80-208Б-18
Преподаватель:	Журавлёв А.А
Оценка:	
Дата:	

Москва 2019

1. Код программы на языке C++
1.1 vertex.h

```
#ifndef OOP_LAB5_VERTEX_H
#define OOP_LAB5_VERTEX_H

#include <iostream>
#include <type_traits>
#include <cmath>


template<class T>
struct vertex {
    T x;
    T y;
    vertex<T>& operator=(vertex<T> A);
};

template<class T>
std::istream& operator>>(std::istream& is, vertex<T>& p) {
    is >> p.x >> p.y;
    return is;
}

template<class T>
std::ostream& operator<<(std::ostream& os, vertex<T> p) {
    os << '(' << p.x << ' ' << p.y << ')';
    return os;
}

template<class T>
vertex<T> operator+(const vertex<T>& A, const vertex<T>& B) {
    vertex<T> res;
    res.x = A.x + B.x;
    res.y = A.y + B.y;
    return res;
}

template<class T>
vertex<T>& vertex<T>::operator=(const vertex<T> A) {
    this->x = A.x;
    this->y = A.y;
    return *this;
}

template<class T>
vertex<T> operator+=(vertex<T> &A, const vertex<T> &B) {
    A.x += B.x;
    A.y += B.y;
    return A;
}
```

```
template<class T>
vertex<T> operator/=(vertex<T>& A, const double B) {
   A.x /= B;
   A.y /= B;
}

template<class T>
double vert_length(vertex<T>& A, vertex<T>& B) {
   double res = sqrt( pow(B.x - A.x, 2) + pow(B.y - A.y, 2) );
   return res;
}

template<class T>
struct is_vertex : std::false_type {};

template<class T>
struct is_vertex<vertex<T>> : std::true_type {};

#endif //OOP_LAB5_VERTEX_H
```

1.2 figure.cpp

```
#ifndef OOP_LAB5_OCTAGON_H
#define OOP_LAB5_OCTAGON_H

#include "vertex.h"

template <class T>
class Octagon {
public:
   vertex<T> dots[8];

   explicit Octagon<T>(std::istream& is) {
      for (auto & dot : dots) {
         is >> dot;
      }
   }

   Octagon<T>() = default;

   double Area() {
      double res = 0;
      for (size_t i = 0; i < 7; i++) {
         res += (dots[i].x * dots[i+1].y) - (dots[i+1].x * dots[i].y);
      }
      res = res + (dots[7].x * dots[0].y) - (dots[0].x * dots[7].y);
      return std::abs(res)/ 2;
   }

   void Printout(std::ostream& os) {
      for (int i = 0; i < 8; ++i) {
         os << this->dots[i];
         if (i != 7) {
```

```
            os << ", ";
        }
    }
    os << std::endl;
}

void operator<< (std::ostream& os) {
    for (int i = 0; i < 8; ++i) {
        os << this->dots[i];
        if (i != 7) {
            os << ", ";
        }
    }
}
};

#endif //OOP_LAB5_OCTAGON_H
```

1.3 queue.h

```
#ifndef OOP_EXERCISE_05_QUEUE_H
#define OOP_EXERCISE_05_QUEUE_H

#include <iterator>
#include <memory>

namespace containers {
    //namespace
    //
    template<class T>
    class queue {
    private:
        struct element;
        size_t size = 0;
    public:
        queue() = default;

        class forward_iterator {
        public:
            using value_type = T;
            using reference = T&;
            using pointer = T*;
            using difference_type = std::ptrdiff_t;
            using iterator_category = std::forward_iterator_tag;
            explicit forward_iterator(element* ptr);
            T& operator*();
            forward_iterator& operator++();
            forward_iterator operator++(int);
            bool operator== (const forward_iterator& other) const;
            bool operator!= (const forward_iterator& other) const;
```

```
        private:
            element* it_ptr;
            friend queue;
        };

        forward_iterator begin();
        forward_iterator end();
        void push(const T& value);
        T& top();
        void pop();
        size_t length();
        void delete_by_it(forward_iterator d_it);
        void delete_by_number(size_t N);
        void insert_by_it(forward_iterator ins_it, T& value);
        void insert_by_number(size_t N, T& value);
        queue& operator=(queue& other);
    private:
        struct element {
            T value;
            std::unique_ptr<element> next_element = nullptr;
            forward_iterator next();
        };

        static std::unique_ptr<element> push_impl(std::unique_ptr<element> cur, const T& value);
        std::unique_ptr<element> first = nullptr;
    };//=========================end-of-class-queue=======================//

    template<class T>
    typename queue<T>::forward_iterator queue<T>::begin() {
        if (size == 0) {
            return forward_iterator(nullptr);
        }
        return forward_iterator(first.get());
    }

    template<class T>
    typename queue<T>::forward_iterator queue<T>::end() {
        return forward_iterator(nullptr);
    }
//================base-methods-of-queue=================================//
    template<class T>
    size_t queue<T>::length() {
      return size;
    }

    template<class T>
    void queue<T>::push(const T& value) {
      first = push_impl(std::move(first), value);
      size++;
    }

    template<class T>
```

```
    std::unique_ptr<typename queue<T>::element> queue<T>::push_impl(std::unique_ptr<element>
cur, const T& value) {
        if (cur != nullptr) {
            cur->next_element = push_impl(std::move(cur->next_element), value);
            return cur;
        }
        return std::unique_ptr<element>(new element{value});
    }

    template<class T>
    void queue<T>::pop() {
        if (size == 0) {
            throw std::logic_error ("can`t pop from empty queue");
        }
        first = std::move(first->next_element);
        size--;
    }

    template<class T>
    T& queue<T>::top() {
        if (size == 0) {
            throw std::logic_error ("queue is empty, lol, it has no top");
        }
        return first->value;
    }
//================advanced-methods==============================//
    template<class T>
    queue<T>& queue<T>::operator=(queue<T>& other){
        size = other.size;
        first = std::move(other.first);
    }

    template<class T>
    void queue<T>::delete_by_it(containers::queue<T>::forward_iterator d_it) {
        queue<T> res;
        int flag = 0;
        T tmp;
        forward_iterator i = this->begin(), end = this->end();
        while(i != end) {
            if (i == d_it) {
                ++i;
                flag = 1;
                this->pop();
            } else {
                ++i;
                tmp = this->top();
                res.push(tmp);
                this->pop();
            }
        }
        if (!flag) throw std::logic_error ("no such element in queue");
        *this = res;
```

```
    }

    template<class T>
    void queue<T>::delete_by_number(size_t N) {
        forward_iterator it = this->begin();
        for (size_t i = 0; i < N+1; ++i) {
            if (i == N) break;
            ++it;
        }
        this->delete_by_it(it);
    }

    template<class T>
    void queue<T>::insert_by_it(containers::queue<T>::forward_iterator ins_it, T& value) {
        queue<T> res;
        int flag = 0;
        T tmp;
        forward_iterator i = this->begin(), end = this->end();
        do {
            if ((i == end) && (i == ins_it)) {
                res.push(value);
                flag = 1;
            } else if (i == ins_it) {
                ++i;
                flag = 1;
                res.push(value);
                tmp = this->top();
                res.push(tmp);
                this->pop();
            } else {
                ++i;
                tmp = this->top();
                res.push(tmp);
                this->pop();
            }
        }  while(i != end);
        if (!flag) throw std::logic_error ("out of queue borders");
        *this = res;
    }

    template<class T>
    void queue<T>::insert_by_number(size_t N, T& value) {
        forward_iterator it = this->begin();
        for (size_t i = 0; i <N+1; ++i) {
            if (i == N) break;
            ++it;
        }
        this->insert_by_it(it, value);
    }
//======================iterator`s-stuff================================//
    template<class T>
    typename queue<T>::forward_iterator queue<T>::element::next() {
```

```
        return forward_iterator(this->next_element.get());
    }

    template<class T>
    queue<T>::forward_iterator::forward_iterator(containers::queue<T>::element *ptr) {
        it_ptr = ptr;
    }

    template<class T>
    T& queue<T>::forward_iterator::operator*() {
        return this->it_ptr->value;
    }

    template<class T>
    typename queue<T>::forward_iterator& queue<T>::forward_iterator::operator++() {
        *this = it_ptr->next();
        return *this;
    }

    template<class T>
    typename queue<T>::forward_iterator queue<T>::forward_iterator::operator++(int) {
        forward_iterator old = *this;
        ++*this;
        return old;
    }

    template<class T>
    bool queue<T>::forward_iterator::operator==(const forward_iterator& other) const {
        return it_ptr == other.it_ptr;
    }

    template<class T>
    bool queue<T>::forward_iterator::operator!=(const forward_iterator& other) const {
        return it_ptr != other.it_ptr;
    }

    //
    //namespace
}



#endif //OOP_EXERCISE_05_QUEUE_H
```

1.4 main.cpp

```
#include <iostream>
#include <algorithm>
#include "octagon.h"
#include "containers/queue.h"

int main() {
```

```
    size_t N;
    float S;
    char option = '0';
    containers::queue<Octagon<int>> q;
    Octagon<int> oct{};
    while (option != 'q') {
        std::cout << "choose option (m for man, q to quit)" << std::endl;
        std:: cin >> option;
        switch (option) {
            case 'q':
                break;
            case 'm':
                std::cout << "1) push new element into queue\n"
                << "2) insert element into chosen position\n"
                << "3) pop element from the queue\n"
                << "4) delete element from the chosen position\n"
                << "5) print queue\n"
                << "6) count elements with area less then chosen value\n" << std::endl;
                break;
            case '1': {
                std::cout << "enter octagon (have to enter dots consequently): " << std::endl;
                oct = Octagon<int>(std::cin);
                q.push(oct);
                break;
            }
            case '2': {
                std::cout << "enter position to insert to: ";
                std::cin >> N;
                std::cout << "enter octagon: ";
                oct = Octagon<int>(std::cin);
                q.insert_by_number(N, oct);
                break;
            }
            case '3': {
                q.pop();
                break;
            }
            case '4': {
                std::cout << "enter position to delete: ";
                std::cin >> N;
                q.delete_by_number(N);
                break;
            }
            case '5': {
                std::for_each(q.begin(), q.end(), [](Octagon<int> &X) { X.Printout(std::cout); });
                break;
            }
            case '6': {
                std::cout << "enter max area to search to: ";
                std::cin >> S;
                std::cout <<"number of elements with value less than " << S << " is " <<
std::count_if(q.begin(), q.end(), [=](Octagon<int>& X){return X.Area() < S;}) << std::endl;
```

```
            break;
          }
          default:
            std::cout << "no such option. Try m for man" << std::endl;
            break;
      }
   }
}
```

2. Ссылка на репозиторий на GitHub

https://github.com/vitalouivi/oop_exercise_05

3. Набор тестов

```
1)
m
1
1 2 3 4 6 2 3 0
5
1
0 0    1 0    2 0    2 1
5
2
2
2 2    1 2    0 2    0 1
5
4
2
5
Q
2)
m
1
1 3 4 6  2 3  0 5
5
1
0 0    1 0    2 0    2 1
5
4
1
4
0
5
Q
```

4. Результаты тестов

```
 choose option (m to open man, q to quit)
m
1) push new element into queue
2) insert element into chosen position
3) pop element from the queue
4) delete element from the chosen position
5) print queue
6) count elements with area less then chosen value

choose option (m to open man, q to quit)
1
enter rectangle (have to enter dots consequently):
1 2
3 4
6 2
3 0
choose option (m to open man, q to quit)
5
(1 2), (3 4), (6 2), (3 0)
choose option (m to open man, q to quit)
1
enter rectangle (have to enter dots consequently):
0 0     1 0     2 0     2 1
choose option (m to open man, q to quit)
5
(1 2), (3 4), (6 2), (3 0)
(0 0), (1 0), (2 0), (2 1)
choose option (m to open man, q to quit)
2
enter position to insert to: 2
enter rectangle: 2 2     1 2     0 2     0 1
choose option (m to open man, q to quit)
5
(1 2), (3 4), (6 2), (3 0)
(2 2), (1 2), (0 2), (0 1)
(0 0), (1 0), (2 0), (2 1)
choose option (m to open man, q to quit)
4
enter position to delete: 2
choose option (m to open man, q to quit)
5
(1 2), (3 4), (6 2), (3 0)
(0 0), (1 0), (2 0), (2 1)
choose option (m to open man, q to quit)
q

C:\Users\Пользователь\Desktop\oop_exercise_05-master\out\build\x64-Debug\oop_exe
```

```
rcise_05.exe (процесс 14724) завершил работу с кодом 0.

choose option (m to open man, q to quit)
m
1) push new element into queue
2) insert element into chosen position
3) pop element from the queue
4) delete element from the chosen position
5) print queue
6) count elements with area less then chosen value

choose option (m to open man, q to quit)
1
enter rectangle (have to enter dots consequently):
1 3 4 6  2 3  0 5
choose option (m to open man, q to quit)
5
(1 3), (4 6), (2 3), (0 5)
choose option (m to open man, q to quit)
1
enter rectangle (have to enter dots consequently):
0 0     1 0     2 0     2 1
choose option (m to open man, q to quit)
5
(1 3), (4 6), (2 3), (0 5)
(0 0), (1 0), (2 0), (2 1)
choose option (m to open man, q to quit)
4
enter position to delete: 1
choose option (m to open man, q to quit)
4
enter position to delete: 0
choose option (m to open man, q to quit)
5
choose option (m to open man, q to quit)
Q
no such option. Try m for man
choose option (m to open man, q to quit)
q

C:\Users\Пользователь\Desktop\oop_exercise_05-master\out\build\x64-Debug\oop_exe
rcise_05.exe (процесс 16684) завершил работу с кодом 0.
```

5. Объяснение работы программы

Коллекция очереди находится в пространсве имён containers.

Private-члены и методы: size – хранит размер очереди (просто для красоты), struct element – описание структуры элемента очереди, first – указатель на первый элемент очереди

Public-члены и методы: push – метод добавления элемента в конец очереди, push_impl – вспомогательный метод для push, top – возвращает первый элемент очереди, pop – выкидывает первый элемент из очереди, length – возвращает размер очереди (опять же для соответствия спецификации очереди), delete_by_it – удаление элемента по итератору, delete_by_number – по номеру создаёт нужный итератор и удаляет по нему, insert_by_it и insert_by_number – аналогично, operator= - перемещает очередь (нужно для вставки/удаления произвольного элемента очереди), класс forward-iterator – задаёт итератор типа forward_iterator со всеми необходимыми методами и типами данных, queue() - дефолтный конструктор.

delete_by_it

Проходит по итераторам всю очередь и перекидывает элемент в новую. Если итератор совпал с удаляемым, то в новую очередь элемент не закидывается, и выставляется флаг успешного удаления.

insert_by_it

Аналогично удалению проходит всю очередь, и при натыкании на нужный итератор закидывает в новую очередь вставляемый элемент.

Вывод

Проделав работу, я изучил принципы работы unique_ptr и узнал о некоторых особенностях их работы. Например, принцип единоличного владения объектом не позволил мне хранить в очереди указатель на последний элемент, что несколько усложнило операцию вставки. Также я узнал, как “правильно” писать итераторы, чтобы они были совместимы со стандартными алгоритмами работы с коллекциями. Поверхностно ознакомился с лямбда-функциями и создал своё первое пространство имён.