Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 5: «Основы работы с коллекциями: Итераторы»

Группа:	М8О-208Б-18, №22
Студент:	Рыженко Иван Александрович
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	11.01.2020

Москва, 2020

1. Задание

Собрать шаблон динамической коллекции согласно варианту задания.
Вариант 22: Пятиугольник. Очередь.

2. TestCases

Test 1.//проверка добавления
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
1
Coordinates of 1 vertex:
Coordinate 'x': 21
Coordinate 'y': 12
Coordinates of 2 vertex:
Coordinate 'x': 124
Coordinate 'y': 12
Coordinates of 3 vertex:
Coordinate 'x': 214
Coordinate 'y': 12
Coordinates of 4 vertex:
Coordinate 'x': 214
Coordinate 'y': 12
Coordinates of 5 vertex:
Coordinate 'x': 241
Coordinate 'y': 12
(21 12),(124 12),(214 12),(214 12),(241 12)
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
6
Input the index
2
Input coordinates of the pentagon
Coordinates of 1 vertex:
Coordinate 'x': 21
Coordinate 'y': 12
Coordinates of 2 vertex:
Coordinate 'x':
12
Coordinate 'y': 12
Coordinates of 3 vertex:
Coordinate 'x': 21
Coordinate 'y': 214
Coordinates of 4 vertex:
Coordinate 'x': 12
Coordinate 'y': 12

Coordinates of 5 vertex:
Coordinate 'x': 214
Coordinate 'y': 12
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
4
Element of queue:
(21 12),(124 12),(214 12),(214 12),(241 12)

Element of queue:
(21 12),(12 12),(21 214),(12 12),(214 12)

1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
1
Coordinates of 1 vertex:
Coordinate 'x': 123
Coordinate 'y': 12
Coordinates of 2 vertex:
Coordinate 'x': 12
Coordinate 'y': 23
Coordinates of 3 vertex:
Coordinate 'x': 12
Coordinate 'y': 12
Coordinates of 4 vertex:
Coordinate 'x': 12
Coordinate 'y': 123
Coordinates of 5 vertex:
Coordinate 'x': 123
Coordinate 'y': 12
(123 12),(12 23),(12 12),(12 123),(123 12)

Test 2.//проверка удаления
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
2

Empty collection.
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
1
Coordinates of 1 vertex:
Coordinate 'x': 12
Coordinate 'y': 12
Coordinates of 2 vertex:
Coordinate 'x': 12
Coordinate 'y': 24
Coordinates of 3 vertex:
Coordinate 'x': 12
Coordinate 'y': 12
Coordinates of 4 vertex:
Coordinate 'x': 412
Coordinate 'y': 124
Coordinates of 5 vertex:
Coordinate 'x': 12
Coordinate 'y': 12
(12 12),(12 24),(12 12),(412 124),(12 12)
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
2
1. Delete the top element of queue
2. Delete figure by index
1
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
4
Empty collection.

Test 3.//работа с пустой коллекцией
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue

```
5. Output the number of elements with area more then input
6. Add figure by index
2
Empty collection.
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
3
Empty collection.
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
4
Empty collection.
1. Add figure in queue
2. Delete element of queue
3. Output element of queue
4. Output all elements of queue
5. Output the number of elements with area more then input
6. Add figure by index
5
Input the area:
12
0
```

3. Адрес репозитория на GitHub

https://github.com/THEproVANO/oop_exercise_05

4. Код программы на C++

Vertex.h

```
#pragma once
#include<iostream>
#include<type_traits>
#include<cmath>

//????? "???????"
template<class T>
struct Vertex {
    using vertex = std::pair<T,T>;
    vertex coordinates;
    Vertex<T>& operator=(Vertex<T> A);
    Vertex() = default;//??????????? ?? ????????
};

template<class T>
std::istream& operator>>(std::istream& is, Vertex<T>& p) {
    std::cout << "Coordinate 'x': ";
```

```
    is >> p.coordinates.first;
    std::cout << "Coordinate 'y': ";
    is >> p.coordinates.second;
        return is;
}

template<class T>
std::ostream& operator<<(std::ostream& os, Vertex <T> p) {
    os << '(' << p.coordinates.first << ' ' << p.coordinates.second << ')';
        return os;
}

template<class T>
Vertex<T> operator+(const Vertex<T> A, const Vertex<T> B) {
    Vertex<T> res;
    res.coordinates.first = A.coordinates.first + B.coordinates.first;
    res.coordinates.second = A.coordinates.second + B.coordinates.second;
        return res;
}

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

template<class T>
Vertex<T> operator+=(Vertex<T> A, const Vertex<T> B) {
    A.coordinates.first += B.coordinates.first;
    A.coordinates.second += B.coordinates.second;
        return A;
}

template<class T>
Vertex<T> operator/=(Vertex<T> A, const double B) {
    A.coordinates.first /= B;
    A.coordinates.second /= B;
}

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

template<class T>
struct is_Vertex<Vertex<T>> : std::true_type {};
```

Pentagon.h

```
#pragma once
#include"vertex.h"

//???????????? ???? ? ???????? ?????
template<class T>
class Pentagon {
public:
    Vertex<T> vertices[5];
        Pentagon() = default;
        Pentagon(std::istream& in);
        void Read(std::istream& in);
        double Area() const;
        void Print(std::ostream& os) const;
        friend std::ostream& operator<< (std::ostream& out, const Pentagon<T>&
point);
};


template<class T>
Pentagon<T>::Pentagon(std::istream& is)//??????????? ?????? "Pentagon"
{
    for (int i = 0; i < 5; i++)
    {
        std::cout << "Coordinates of " << i+1 << " vertex: \n";
                is >> this->vertices[i];
        }
}

        template<class T>
        double Pentagon<T>::Area() const {
                double Area = 0;
                for (int i = 0; i < 5; i++) {
            Area += (vertices[i].coordinates.first) * (vertices[(i + 1) %
5].coordinates.second) - (vertices[(i + 1) % 5].coordinates.first) * (ver-
tices[i].coordinates.second);
                }
                Area *= 0.5;
                return abs(Area);
```

```
}

template<class T>
void Pentagon<T>::Print(std::ostream& os) const  {
        for (int i = 0; i < 5; i++) {
                os << this->vertices[i];
                if (i != 4) {
                        os << ',';
                }
        }
        os << std::endl;
}

template<class T>
void Pentagon<T>::Read(std::istream& in) {
 for (int i = 0; i < 5; i++)
 {
     std::cout << "Coordinates of " << i+1 << " vertex: \n";
     in >> this->vertices[i];
 }
 }

template<class T>
std::ostream& operator<<(std::ostream& os, const Pentagon<T>& point) {
        for (int i = 0; i < 5; i++) {
                os << point.vertices[i];
                if (i != 5) {
                        os << ',';
                }
        }
}
```

Queue.h

```
#include <iterator>
#include <memory>

namespace Containers {

//????? ????????? "???????"
      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 class Queue;
                   };

                   forward_iterator begin();
                   forward_iterator end();
                   void push(const T& value);
                   T& top();
                   T& bottom();
                   void pop();
                   size_t length();
                   void delete_by_it(forward_iterator d_it);
         void delete_by_index(size_t N);//????????? ?? ???????
                   void insert_by_it(forward_iterator ins_it, T& value);
         void insert_by_index(size_t N, T& value);//??????????? ?? ???????
         Queue& operator=(Queue& other);
        private:
         struct element
         {
```

```
                    T value;
                    std::unique_ptr<element> next_element = nullptr;
                    forward_iterator next();
                };

        void push_impl(std::unique_ptr<element>& cur, const T& value);
        std::unique_ptr<element> pop_impl(std::unique_ptr<element> cur);
                std::unique_ptr<element> first = nullptr;
        };


        template<class T>
    typename Queue<T>::forward_iterator Queue<T>::begin() {
                return forward_iterator(first.get());
        }

        template<class T>
    typename Queue<T>::forward_iterator Queue<T>::end() {
                return forward_iterator(nullptr);
        }



        template<class T>
    size_t Queue<T>::length()
{//?????????? ????? ?????????, ?????????? ? ?????????
        return this->size;
        }

        template<class T>
    void Queue<T>::push(const T& value)//?????????? ???????? ? ?????????
    {
        push_impl(this->first, value);
                size++;
        }

        template<class T>
    void Queue<T>::push_impl(std::unique_ptr<element>& cur, const T& value)
    {
        if (cur == nullptr)
        {
            cur = std::unique_ptr<element>(new element{value});
            return;
        }
        else
            push_impl(cur->next_element, value);
    }

    template<class T>
    void Queue<T>::pop()
    {
        if (size == 0)
            throw std::logic_error("Queue is empty");
        first = std::move(first->next_element);
                size--;
        }


        template<class T>
    std::unique_ptr<typename Queue<T>::element>
Queue<T>::pop_impl(std::unique_ptr<element> cur)
    {
        if (cur->next_element != nullptr)
        {
                    cur->next_element = pop_impl(std::move(cur->next_ele-
ment));
                    return cur;
                }
        else
            return nullptr;
        }

        template<class T>
    T& Queue<T>::bottom()
    {
        if (size == 0)
            throw std::logic_error("Queue is empty");
                forward_iterator i = this->begin();
                while (i.it_ptr->next() != this->end()) {
                    i++;
                }
                return *i;
        }

        template<class T>
    T& Queue<T>::top()
    {
                return first->value;
```

```
        }

        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)
    {
            forward_iterator i = this->begin(), end = this->end();
        if (d_it == end)
            throw std::logic_error("Out of borders");
        if (d_it == this->begin())
        {
                std::unique_ptr<element> tmp;
                tmp = std::move(first->next_element);
                first = std::move(tmp);
                return;
            }
            while ((i.it_ptr != nullptr) && (i.it_ptr->next() != d_it)) {
                ++i;
            }
        if (i.it_ptr == nullptr) throw std::logic_error("Out of borders");
            i.it_ptr->next_element = std::move(d_it.it_ptr->next_element);
            size--;
        }

        template<class T>
    void Queue<T>::delete_by_index(size_t N)
    {
        if (N >= this->size)
            return;
            forward_iterator it = this->begin();
        for (size_t i = 0; i < N; ++i)
                ++it;
            this->delete_by_it(it);
        }

        template<class T>
    void Queue<T>::insert_by_it(Containers::Queue<T>::forward_iterator
ins_it, T& value)
    {
            auto tmp = std::unique_ptr<element>(new element{ value });
            forward_iterator i = this->begin();
            if (ins_it == this->begin()) {
                tmp->next_element = std::move(first);
                first = std::move(tmp);
                size++;
                return;
            }
            while ((i.it_ptr != nullptr) && (i.it_ptr->next() != ins_it))
{
                ++i;
            }
        if (i.it_ptr == nullptr) throw std::logic_error("Out of borders");
            tmp->next_element = std::move(i.it_ptr->next_element);
            i.it_ptr->next_element = std::move(tmp);
            size++;
        }


        template<class T>
    void Queue<T>::insert_by_index(size_t N, T& value) {
            forward_iterator it = this->begin();
            if (N >= this->length())
                it = this->end();
            else
            for (size_t i = 1; i <= N; ++i) {
                ++it;
            }
            this->insert_by_it(it, value);
        }

        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>::ele-
ment* 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::opera-
tor++() {
        if (it_ptr == nullptr) throw std::logic_error("Out of queue");
            *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;
        }

}
```

Main.cpp

```
#include <iostream>
#include <algorithm>
#include <locale.h>
#include "Pentagon.h"
#include "queue.h"

void Menu1()
{
    std::cout << "1. Add figure in queue\n";
    std::cout << "2. Delete element of queue\n";
    std::cout << "3. Output element of queue\n";
    std::cout << "4. Output all elements of queue\n";
    std::cout << "5. Output the number of elements with area more then input\
n";
    std::cout << "6. Add figure by index\n";
}

void DeleteMenu()
{
    std::cout << "1. Delete the top element of queue\n";
    std::cout << "2. Delete figure by index\n";
}

void PrintMenu()
{
    std::cout << "1. Output the top element\n";
    std::cout << "2. Output the last element\n";
}

int main()
{
    Containers::Queue<Pentagon<int>> Myqueue;

        Pentagon<int> TempPentagon;

        while (true) {
            Menu1();
        int n, m;
            size_t ind;
            double s;
            std::cin >> n;
        switch (n)
        {
            case 1:
            TempPentagon.Read(std::cin);
            TempPentagon.Print(std::cout);
            Myqueue.push(TempPentagon);
                    break;
            case 2:
            if (Myqueue.length() == 0)
```

```
            {
                std::cout << "Empty collection.\n";
                break;
            }
                    DeleteMenu();
                    std::cin >> m;
                    switch (m) {
                    case 1:
                Myqueue.pop();
                        break;
                    case 2:
                std::cout << "Input the index: ";
                        std::cin >> ind;
                if (ind > Myqueue.length())
                {
                    std::cout << "Index is out of bourders.\n";
                    break;
                }
                Myqueue.delete_by_index(ind);
                        break;
                    default:
                        break;
                    }
                    break;
               case 3:
            if (Myqueue.length() == 0)
            {
                std::cout << "Empty collection.\n";
                break;
            }
                    PrintMenu();
                    std::cin >> m;
                    switch (m) {
                    case 1:
                Myqueue.top().Print(std::cout);
                        std::cout << std::endl;
                        break;

                    case 2:
                Myqueue.bottom().Print(std::cout);
                        std::cout << std::endl;
                        break;

                    default:
                std::cout << "Incorrect input\n";
                        break;
                    }
                    break;
               case 4:
            if (Myqueue.length() == 0)
            {
                std::cout << "Empty collection.\n";
                break;
            }
            std::for_each(Myqueue.begin(), Myqueue.end(), [](Pentagon<int>&
X) { std::cout << "Element of queue:\n"; X.Print(std::cout); std::cout <<
std::endl; });
                    break;
               case 5:
            std::cout << "Input the area:\n";
                    std::cin >> s;
            std::cout << std::count_if(Myqueue.begin(), Myqueue.end(), [=]
(Pentagon<int>& X) {return X.Area() > s; }) << std::endl;
                    break;
               case 6:
            std::cout << "Input the index\n";
                    std::cin >> ind;
            std::cout << "Input coordinates of the pentagon\n";
                    TempPentagon.Read(std::cin);
            Myqueue.insert_by_index(ind, TempPentagon);
                    break;
               default:
                    return 0;
                }
        }
        return 0;
}
```

5. Объяснение результатов работы программы

При запуске программы в консоль выводится меню:

“Add figure in queue” – функция добавления элемента в очередь.

“Delete element of queue” – удаление верхнего элемента очереди (аналог pop) или удаление элемента по индексу

“Output element of queue” – вывод в консоль первого или последнего элемента коллекции.

“Output all elements of queue” – вывод всех элементов коллекции в консоль. (Если коллекция пуста – выводится “Empty queue”)

“Output the number of elements with area more than input” – вывод элементов коллекции с площадью больше данной.

“Add figure by index” – добавление фигуры по заданному индексу. (Если индекс превосходит размер коллекции – элемент добавляется в конец)

6. Вывод

В данной работе была реализована коллекция “Очередь”, а также итератор к ней, позволяющий передвигаться по её элементам.