Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Основы работы с коллекциями : итераторы
Цель работы :
изучение основ работы с коллекциями, знакомство с итераторами

Студент:	Голубев В.С.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	4
Оценка:	
Дата:	

Москва, 2020

1. Код программы на языке C++:

Point.h

```
#ifndef LAB5_POINT_H
 #define LAB5_POINT_H
 #include <iostream>
 #include <algorithm>
 #include "cmath"
 template<class T>
 struct point {
     T x;
     T y;
     double VectNorm(point l, point r) const;
     double VectProd(point l, point r) const;
 };
 template<class T>
 std::istream& operator>> (std::istream& is, point<T>& p) {
     is >> p.x >> p.y;
     return is;
 }
 template<class T>
 std::ostream& operator<< (std::ostream& os, const point<T>& p) {
     os << p.x << ' ' << p.y;
     return os;
 }
 template<class T>
 point<T> operator-(const point<T> l, const point<T> r) {
     point<T> res;
     res.x = r.x - l.x;
     res.y = r.y - l.y;
     return res;
 }
 template<class T>
 double VectProd(point<T> l, point<T> r) {
     return std::abs(l.x * r.y - r.x * l.y);
 }
 template<class T>
 double VectNorm(point<T> l, point<T> r) {
     point<T> vect = operator-(l,r);
     double res = sqrt(vect.x* vect.x + vect.y * vect.y);
     return res;
 }
 #endif //LAB5_POINT_H
```

Stack.h

```
#ifndef LAB5_STACK_H
#define LAB5_STACK_H
#include <iterator>
#include <memory>
namespace countainer {
    template<class T>
    class Stack {
    private:
        class StackNode;
    public:
        class ForwardIterator {
        public:
            using value_type = T;
            using reference = T &;
            using pointer = T *;
            using difference_type = ptrdiff_t;
            using iterator_category = std::forward_iterator_tag;
            ForwardIterator(StackNode *node) : Ptr(node) {};
            T &operator*();
            ForwardIterator &operator++();
            ForwardIterator operator++(int);
            bool operator==(const ForwardIterator &it) const;
            bool operator!=(const ForwardIterator &it) const;
        private:
            StackNode *Ptr;
            friend class Stack;
        };
        ForwardIterator begin();
        ForwardIterator end();
        T& Top();
        void Insert(int& index,const T &value );
        void InsertHelp(const ForwardIterator &it, const T &value);
        void Erase(const ForwardIterator &it);
        void Pop();
        void Push(const T &value);
        Stack() = default;
        Stack(const Stack &) = delete;
        Stack &operator=(const Stack &) = delete;
        size_t Size = 0;
    private:
        class StackNode {
        public:
            T Value;
            std::unique_ptr<StackNode> NextNode = nullptr;
            ForwardIterator next();
            StackNode(const T &value, std::unique_ptr<StackNode> next) :
```

```
Value(value), NextNode(std::move(next)) {};
        };
        std::unique_ptr<StackNode> Head = nullptr;
    };
    template<class T>
    typename Stack<T>::ForwardIterator Stack<T>::StackNode::next() {
        return {NextNode.get()};
    }
    template<class T>
    T &Stack<T>::ForwardIterator::operator*() {
        return Ptr->Value;
    }
    template<class T>
    typename Stack<T>::ForwardIterator
&Stack<T>::ForwardIterator::operator++() {
        *this = Ptr->next();
        return *this;
    }
    template<class T>
    typename Stack<T>::ForwardIterator
Stack<T>::ForwardIterator::operator++(int) {
        ForwardIterator prev = *this;
        ++(*this);
        return prev;
    }
    template<class T>
    bool Stack<T>::ForwardIterator::operator!=(const ForwardIterator
&other) const {
        return Ptr != other.Ptr;
    }
    template<class T>
    bool Stack<T>::ForwardIterator::operator==(const ForwardIterator
&other) const {
        return Ptr == other.Ptr;
    }
    template<class T>
    typename Stack<T>::ForwardIterator Stack<T>::begin() {
        return Head.get();
    }
    template<class T>
    typename Stack<T>::ForwardIterator Stack<T>::end() {
        return nullptr;
    }
    template<class T>
    void Stack<T>::Insert(int &index, const T &value) {
        int id = index - 1;
        if (index < 0 || index > this->Size) {
            throw std::logic_error("Out of bounds\n");
```

```
        }
        if (id == -1) {
            this->Push(value);
        }
        else {
            auto it = this->begin();
            for (int i = 0; i < id; ++i) {
                ++it;
            }
            this->InsertHelp(it,value);
        }
    }
    template<class T>
    void Stack<T>::InsertHelp(const ForwardIterator &it, const T &value)
{
        std::unique_ptr<StackNode> newNode(new StackNode(value,
nullptr));
        if (it.Ptr == nullptr && Size != 0) {
            throw std::logic_error("Out of bounds");
        }
        if (Size == 0) {
            Head = std::move(newNode);
            ++Size;
        } else {
            newNode->NextNode = std::move(it.Ptr->NextNode);
            it.Ptr->NextNode = std::move(newNode);
            ++Size;
        }
    }
    template<class T>
    void Stack<T>::Push(const T &value) {
        std::unique_ptr<StackNode> newNode(new StackNode(value,
nullptr));
        newNode->NextNode = std::move(Head);
        Head = std::move(newNode);
        ++Size;
    }
    template<class T>
    T& Stack<T>::Top() {
        if (Head.get()) {
            return Head->Value;
        } else {
            throw std::logic_error("Stack is empty");
        }
    }
    template<class T>
    void Stack<T>::Pop() {
        if (Head.get() == nullptr) {
```

```
            throw std::logic_error("Stack is empty");
        }
        Head = std::move(Head->NextNode);
        --Size;
    }
    template<class T>
    void Stack<T>::Erase(const Stack<T>::ForwardIterator &it) {
        if (it.Ptr == nullptr) {
            throw std::logic_error("Invalid iterator");
        }
        if (it == this->begin()) {
            Head = std::move(Head->NextNode);
        } else {
            auto tmp = this->begin();
            while (tmp.Ptr->next() != it.Ptr) {
                ++tmp;
            }
            tmp.Ptr->NextNode = std::move(it.Ptr->NextNode);
        }
    }
}
#endif //LAB5_STACK_H
```

Trapeze.h

```
#ifndef LAB5_TRAPEZE_H
 #define LAB5_TRAPEZE_H
 #include "point.h"
  template <class T>
 struct trapeze {
 private:
     point<T> a_,b_,c_,d_;
 public:
     point<T> center() const;
     void print(std::ostream& os) const ;
     double area() const;
     trapeze(std::istream& is);
 };
 template<class T>
 double trapeze<T>::area() const {
     if ( (VectProd(operator-(a_,b_), operator-(c_,d_)) == 0) &&
 (VectProd(operator-(b_,c_), operator-(a_,d_)) == 0) ) {
         return fabs((VectProd(operator-(a_,b_), operator-(a_,d_)))) ;
     } else if (VectProd(operator-(a_,b_), operator-(d_,c_)) == 0) {
         return ((VectNorm(a_, b_) + VectNorm(d_, c_)) / 2) * sqrt(
                 VectNorm(d_, a_) * VectNorm(d_, a_) - (
                         pow((
                                     (pow((VectNorm(d_, c_) - VectNorm(a_,
```

```
b_)), 2) +
                                        VectNorm(d_, a_) * VectNorm(d_, a_)
- VectNorm(b_, c_) * VectNorm(b_, c_)) /
                                        (2 * (VectNorm(d_, c_) - VectNorm(a_,
b_)))
                                ), 2)
                )
        );
    } else if (VectProd(operator-(b_,c_), operator-(a_,d_)) == 0) {
        return ((VectNorm(b_, c_) + VectNorm(a_, d_)) / 2) * sqrt(
                VectNorm(a_, b_) * VectNorm(a_, b_) - (
                        pow((
                                        (pow((VectNorm(a_, d_) - VectNorm(b_,
c_)), 2) +
                                        VectNorm(a_, b_) * VectNorm(a_, b_)
- VectNorm(c_, d_) * VectNorm(c_, d_)) /
                                        (2 * (VectNorm(a_, d_) - VectNorm(b_,
c_)))
                                ), 2)
                )
        );
    }
}
template<class T>
trapeze<T>::trapeze(std::istream& is) {
    is >> a_ >> b_ >> c_ >> d_ ;
}
template<class T>
void trapeze<T>::print(std::ostream& os) const {
    os << "coordinate:\n"<<"\n"<< a_ << '\n' << b_ << '\n' << c_ << '\n'
<< d_ << '\n';
}
template<class T>
point<T> trapeze<T>::center() const {
    T x,y;
    x = (a_.x + b_.x + c_.x + d_.x ) / 4;
    y = (a_.y + b_.y + c_.y + d_.y) / 4;
    return {x,y};
}
#endif //LAB5_TRAPEZE_H
```

Main.cpp

```
#include <iostream>
#include "stack.h"
#include "trapeze.h"
#include <algorithm>
void menu() {
    std::cout << "1 - add(1 - push, 2 - insert  by
iterator(enter index new elem)\n"
                 "2 - delete(1 - pop, 2 - delete by
iterator(enter index)\n"
                 "3 - top\n"
                 "4 - print\n"
                 "5 - count if(enter max area)\n"
                 "6 - exit\n";
}
void usingStack() {
    int command, minicommand, index;
    double val;
    countainer::Stack<trapeze<double>> st;
    for (;;) {
        std::cin >> command;
        if (command == 1) {
            std::cin >> minicommand;
            if (minicommand == 1) {
                trapeze<double> p(std::cin);
                st.Push(p);
            } else if (minicommand == 2) {
                std::cin >> index;
                try {
                    trapeze<double> p(std::cin);
                    st.Insert(index,p);
                } catch (std::logic_error &e) {
                    std::cout << e.what() <<
std::endl;
                    continue;
                }
            }
        } else if (command == 6) {
            break;
        } else if (command == 2) {
            std::cin >> minicommand;
            if (minicommand == 1) {
                try {
                    st.Pop();
                } catch (std::logic_error &e) {
                    std::cout << e.what() <<
std::endl;
```

```
                    continue;
                }
            }
            if (minicommand == 2) {
                std::cin >> index;
                try {
                    if (index < 0 || index >
st.Size) {
                        throw std::logic_error("Out
of bounds\n");
                    }
                    auto it = st.begin();
                    for (int i = 0; i < index; ++i)
{
                        ++it;
                    }
                    st.Erase(it);
                }
                catch (std::logic_error &e) {
                    std::cout << e.what() <<
std::endl;
                    continue;
                }
            }
        } else if (command == 3) {
            try {
                st.Top().print(std::cout);
            }
            catch (std::logic_error &e) {
                std::cout << e.what() << std::endl;
                continue;
            }
        } else if (command == 4) {
            for (auto elem: st) {
                elem.print(std::cout);
            }
        } else if (command == 5) {
            std::cin >> val;
            std::cout << std::count_if(st.begin(),
st.end(), [val](trapeze<double> r) { return
r.area() < val; })
                      << std::endl;
        } else {
            std::cout << "Error command\n";
            continue;
        }
    }
}
```

```
int main() {
    menu();
    usingStack();
    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.14)
project(lab5)
set(CMAKE_CXX_STANDARD 17)
add_executable(lab5 main.cpp point.h trapeze.h stack.h)
```

2. Ссылка на репозиторий на GitHub

https://github.com/VSGolubev-bald/oop_exercise_05

3. Вывод :

Выполнив данную лабораторную работу, я обучился азам работы с коллекциями, а также реализовал собственный итератор.

Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Основы работы с коллекциями : итераторы
Цель работы :
изучение основ работы с контейнерами, знакомство с концепцией аллокаторов памяти

Студент:	Голубев В.С.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	4
Оценка:	
Дата:	

Москва, 2020

1. Код программы на языке C++:

my_allocator.h

```
#ifndef LAB6_MY_ALLOCATOR_H
#define LAB6_MY_ALLOCATOR_H
#include <cstdlib>
#include <cstdint>
#include <memory>
#include "queue.h"
namespace my_all {
template <typename T, size_t ALLOC_SIZE>
    class my_allocator {
    public:
        using value_type = T;
        using size_type = std::size_t;
        using difference_type = std::ptrdiff_t;
        using is_always_equal = std::false_type;
        my_allocator(const my_allocator &) = delete;
        my_allocator(my_allocator &&) = delete;
        template<class U>
        struct rebind {
            using other = my_allocator<U, ALLOC_SIZE>;
        };
        my_allocator() {
            size_t object_count = ALLOC_SIZE / sizeof(T);
            memory = reinterpret_cast<char *>(operator new(sizeof(T) *
object_count));
            for (size_t i = 0; i < object_count; ++i) {
                free_blocks.push(memory + sizeof(T) * i);
            }
        }
        ~my_allocator() {
            operator delete(memory);
        }
        T *allocate(size_t size) {
            if (size > 1) {
                throw std::logic_error("no way to allocate");
            }
            if (free_blocks.empty()) {
                throw std::bad_alloc();
            }
            T *temp = reinterpret_cast<T *>(free_blocks.top());
            free_blocks.pop();
            return temp;
        }
        void deallocate(T *ptr, size_t size) {
```

```
            if (size != 1) {
                throw std::logic_error("Can't do that");
            }
            free_blocks.push(reinterpret_cast<char *>(ptr));
        }
    private:
        countainer::queue<char *> free_blocks;
        char *memory;
    };
}
#endif //LAB6_MY_ALLOCATOR_H
```

Stack.h

```
#ifndef LAB5_STACK_H
#define LAB5_STACK_H
#include <iterator>
#include <memory>
namespace countainer {
    template<class T>
    class Stack {
    private:
        class StackNode;
    public:
        class ForwardIterator {
        public:
            using value_type = T;
            using reference = T &;
            using pointer = T *;
            using difference_type = ptrdiff_t;
            using iterator_category = std::forward_iterator_tag;
            ForwardIterator(StackNode *node) : Ptr(node) {};
            T &operator*();
            ForwardIterator &operator++();
            ForwardIterator operator++(int);
            bool operator==(const ForwardIterator &it) const;
            bool operator!=(const ForwardIterator &it) const;
        private:
            StackNode *Ptr;
            friend class Stack;
        };
        ForwardIterator begin();
        ForwardIterator end();
        T& Top();
        void Insert(int& index,const T &value );
        void InsertHelp(const ForwardIterator &it, const T &value);
        void Erase(const ForwardIterator &it);
```

```
        void Pop();
        void Push(const T &value);
        Stack() = default;
        Stack(const Stack &) = delete;
        Stack &operator=(const Stack &) = delete;
        size_t Size = 0;
    private:
        class StackNode {
        public:
            T Value;
            std::unique_ptr<StackNode> NextNode = nullptr;
            ForwardIterator next();
            StackNode(const T &value, std::unique_ptr<StackNode> next) :
Value(value), NextNode(std::move(next)) {};
        };
        std::unique_ptr<StackNode> Head = nullptr;
    };
    template<class T>
    typename Stack<T>::ForwardIterator Stack<T>::StackNode::next() {
        return {NextNode.get()};
    }
    template<class T>
    T &Stack<T>::ForwardIterator::operator*() {
        return Ptr->Value;
    }
    template<class T>
    typename Stack<T>::ForwardIterator
&Stack<T>::ForwardIterator::operator++() {
        *this = Ptr->next();
        return *this;
    }
    template<class T>
    typename Stack<T>::ForwardIterator
Stack<T>::ForwardIterator::operator++(int) {
        ForwardIterator prev = *this;
        ++(*this);
        return prev;
    }
    template<class T>
    bool Stack<T>::ForwardIterator::operator!=(const ForwardIterator
&other) const {
        return Ptr != other.Ptr;
    }
    template<class T>
    bool Stack<T>::ForwardIterator::operator==(const ForwardIterator
&other) const {
        return Ptr == other.Ptr;
    }
```

```
    template<class T>
    typename Stack<T>::ForwardIterator Stack<T>::begin() {
        return Head.get();
    }
    template<class T>
    typename Stack<T>::ForwardIterator Stack<T>::end() {
        return nullptr;
    }
    template<class T>
    void Stack<T>::Insert(int &index, const T &value) {
        int id = index - 1;
        if (index < 0 || index > this->Size) {
            throw std::logic_error("Out of bounds\n");
        }
        if (id == -1) {
            this->Push(value);
        }
        else {
            auto it = this->begin();
            for (int i = 0; i < id; ++i) {
                ++it;
            }
            this->InsertHelp(it,value);
        }
    }
    template<class T>
    void Stack<T>::InsertHelp(const ForwardIterator &it, const T &value)
{
        std::unique_ptr<StackNode> newNode(new StackNode(value,
nullptr));
        if (it.Ptr == nullptr && Size != 0) {
            throw std::logic_error("Out of bounds");
        }
        if (Size == 0) {
            Head = std::move(newNode);
            ++Size;
        } else {
            newNode->NextNode = std::move(it.Ptr->NextNode);
            it.Ptr->NextNode = std::move(newNode);
            ++Size;
        }
    }
    template<class T>
    void Stack<T>::Push(const T &value) {
        std::unique_ptr<StackNode> newNode(new StackNode(value,
nullptr));
        newNode->NextNode = std::move(Head);
        Head = std::move(newNode);
```

```
        ++Size;
    }
    template<class T>
    T& Stack<T>::Top() {
        if (Head.get()) {
            return Head->Value;
        } else {
            throw std::logic_error("Stack is empty");
        }
    }
    template<class T>
    void Stack<T>::Pop() {
        if (Head.get() == nullptr) {
            throw std::logic_error("Stack is empty");
        }
        Head = std::move(Head->NextNode);
        --Size;
    }
    template<class T>
    void Stack<T>::Erase(const Stack<T>::ForwardIterator &it) {
        if (it.Ptr == nullptr) {
            throw std::logic_error("Invalid iterator");
        }
        if (it == this->begin()) {
            Head = std::move(Head->NextNode);
        } else {
            auto tmp = this->begin();
            while (tmp.Ptr->next() != it.Ptr) {
                ++tmp;
            }
            tmp.Ptr->NextNode = std::move(it.Ptr->NextNode);
        }
    }
}
#endif //LAB5_STACK_H
```

Point.h

```
#ifndef LAB6_POINT_H
#include <iostream>
#include <algorithm>
#include "cmath"
template<class T>
struct point {
    T x;
    T y;
    double VectNorm(point l, point r) const;
```

```
    double VectProd(point l, point r) const;
};
template<class T>
std::istream& operator>> (std::istream& is, point<T>& p) {
    is >> p.x >> p.y;
    return is;
}
template<class T>
std::ostream& operator<< (std::ostream& os, const point<T>& p) {
    os << p.x << ' ' << p.y;
    return os;
}
template<class T>
point<T> operator-(const point<T> l, const point<T> r) {
    point<T> res;
    res.x = r.x - l.x;
    res.y = r.y - l.y;
    return res;
}
template<class T>
double VectProd(point<T> l, point<T> r) {
    return std::abs(l.x * r.y - r.x * l.y);
}
template<class T>
double VectNorm(point<T> l, point<T> r) {
    point<T> vect = operator-(l,r);
    double res = sqrt(vect.x* vect.x + vect.y * vect.y);
    return res;
}
#endif //LAB6_POINT_H
```

queue.h

```
    #ifndef LAB6_QUEUE_H
    #define LAB6_QUEUE_H
    #ifndef QUEUE_H
    #define QUEUE_H
    #include <memory>
    #include <exception>
    #include <cstdint>
    typedef unsigned long long ull;
    namespace countainer {
        template <typename T, typename my_allocator>
        class queue;
        template <typename T>
```

```
    class lst_node;
    template <typename T, typename my_allocator>
    class iterator;
    template <typename T>
    struct lst_node {
        lst_node() = default;
        lst_node(T new_value) : value(new_value) {}
        T value;
        std::shared_ptr<lst_node> next = nullptr;
        std::weak_ptr<lst_node> prev;
    };
    template<typename T, typename my_allocator =
std::allocator<T>>
    class queue {
    public:
        using value_type = T;
        using size_type = ull;
        using reference = value_type&;
        friend iterator<T, my_allocator>;
        using allocator_type = typename my_allocator::template
rebind<lst_node<T>>::other;
        struct deleter {
            deleter(allocator_type* allocator) :
allocator_(allocator) {}
            void operator() (lst_node<T>* ptr) {
                if (ptr != nullptr) {

std::allocator_traits<allocator_type>::destroy(*allocator_,ptr);
                    allocator_->deallocate(ptr, 1);
                }
            }
        private:
            allocator_type* allocator_;
        };
    public:
        queue() {
            lst_node<T>* ptr = allocator_.allocate(1);

std::allocator_traits<allocator_type>::construct(allocator_,
ptr);
            std::shared_ptr<lst_node<T>> new_elem(ptr,
deleter(&allocator_));
            tail_ = new_elem;
            head_ = tail_;
            size_ = 0;
        }
        queue(const queue& q) = delete;
        queue& operator = (const queue&) = delete;
```

```
        void pop() {
            if (empty()) {
                throw std::out_of_range("empty");
            }
            head_ = head_->next;
            size_--;
        }
        reference top() {
            if (empty()) {
                throw std::logic_error("empty");
            }
            return head_->value;
        }
        size_type size() {
            return size_;
        }
        bool empty() {
            return head_ == tail_;
        }
        iterator<T, my_allocator> begin() {
            return iterator<T,my_allocator>(head_, this);
        }
        iterator<T, my_allocator> end() {
            return iterator<T, my_allocator>(tail_, this);
        }
        void push(const T &value) {
            lst_node<T>* ptr = allocator_.allocate(1);

std::allocator_traits<allocator_type>::construct(allocator_,
ptr, value);
            std::shared_ptr<lst_node<T>> new_elem(ptr,
deleter(&allocator_));
            if (empty()) {
                head_ = new_elem;
                head_->next = tail_;
                tail_->prev = head_;
            } else {
                tail_->prev.lock()->next = new_elem;
                new_elem->prev = tail_->prev;
                new_elem->next = tail_;
                tail_->prev = new_elem;
            }
            size_++;
        }
        void it_rmv(iterator<T, my_allocator> it) {
            std::shared_ptr<lst_node<T>> tmp = it.item_.lock();
            if (it == end()) {
                throw std::logic_error("can't remove end
```

```
iterator");
            }
            if (it == begin()) {
                pop();
                return ;
            }
            std::shared_ptr<lst_node<T>> next_tmp = tmp->next;
            std::weak_ptr<lst_node<T>> prev_tmp = tmp->prev;
            prev_tmp.lock()->next = next_tmp;
            next_tmp->prev = prev_tmp;
            size_--;
        }
        void it_insert(iterator<T, my_allocator> it, const T&
value) {
            std::shared_ptr <lst_node<T>> it_ptr =
it.item_.lock();
            if (it == end()) {
                push(value);
                return;
            }
            lst_node<T>* ptr = allocator_.allocate(1);

std::allocator_traits<allocator_type>::construct(allocator_,
ptr, value);
            std::shared_ptr<lst_node<T>> new_elem(ptr,
deleter(&allocator_));
            if (it == begin()) {
                new_elem->next = head_;
                head_->prev = new_elem;
                head_ = new_elem;
                size_++;
                return ;
            }
            std::shared_ptr <lst_node<T>> ptr_next = it_ptr;
            std::weak_ptr <lst_node<T>> ptr_prev = it_ptr ->
prev;
            new_elem->prev = ptr_prev;
            ptr_prev.lock()->next = new_elem;
            new_elem->next = ptr_next;
            ptr_next->prev = new_elem;
            size_++;
        }
    private:
        allocator_type allocator_;
        std::shared_ptr<lst_node<T>> head_;
        std::shared_ptr<lst_node<T>> tail_;
        int size_;
    };
```

```
    template<typename T, typename my_allocator>
    class iterator {
        friend queue<T, my_allocator>;
    public:
        using value_type = T;
        using reference = T&;
        using pointer = T*;
        using difference_type = ptrdiff_t;
        using iterator_category = std::forward_iterator_tag;
        iterator(std::shared_ptr<lst_node<T>> init_ptr,const
queue<T, my_allocator>* ptr) : item_(init_ptr), lst_(ptr) {}
        iterator(const iterator& it) {
            item_ = it.item_;
            lst_ = it.lst_;
        }
        iterator& operator= (const iterator& it) {
            item_ = it.item_;
            return *this;
        }
        iterator& operator++ () {
            std::shared_ptr<lst_node<T>> tmp = item_.lock();
            if (tmp) {
                if (tmp->next == nullptr) {
                    throw std::logic_error("out of bounds");
                }
                tmp = tmp->next;
                item_ = tmp;
                return  *this;
            }
            throw std::logic_error("smt strange");
        }
        iterator operator++ (int) {
            iterator res(*this);
            ++(*this);
            return res;
        }
        reference operator*() {
            return item_.lock()->value;
        }
        pointer operator->() {
            return &item_->value;
        }
        bool operator!= (const iterator& example) {
            return !(*this == example);
        }
        bool operator== (const iterator& example) {
            return item_.lock() == example.item_.lock();
        }
```

```
    private:
        std::weak_ptr<lst_node<T>> item_;
        const queue<T, my_allocator>* lst_;
    };
}
#endif
#endif //LAB6_QUEUE_H
```

Trapeze.h

```
#ifndef LAB6_TRAPEZE_H
#define LAB6_TRAPEZE_H
#include "point.h"
template <class T>
struct trapeze {
private:
    point<T> a_,b_,c_,d_;
public:
    point<T> center() const;
    void print(std::ostream& os) const ;
    double area() const;
    trapeze(std::istream& is);
};
template<class T>
double trapeze<T>::area() const {
    if ( (VectProd(operator-(a_,b_), operator-
(c_,d_)) == 0) && (VectProd(operator-(b_,c_),
operator-(a_,d_)) == 0) ) {
        return fabs((VectProd(operator-(a_,b_),
operator-(a_,d_)))) ;
    } else if (VectProd(operator-(a_,b_),
operator-(d_,c_)) == 0) {
        return ((VectNorm(a_, b_) + VectNorm(d_,
c_)) / 2) * sqrt(
                VectNorm(d_, a_) * VectNorm(d_,
a_) - (
                        pow((

(pow((VectNorm(d_, c_) - VectNorm(a_, b_)), 2) +

VectNorm(d_, a_) * VectNorm(d_, a_) -
VectNorm(b_, c_) * VectNorm(b_, c_)) /
                                    (2 *
(VectNorm(d_, c_) - VectNorm(a_, b_)))
                                ), 2)
                )
        );
    } else if (VectProd(operator-(b_,c_),
```

```
operator-(a_,d_)) == 0) {
        return ((VectNorm(b_, c_) + VectNorm(a_,
d_)) / 2) * sqrt(
                VectNorm(a_, b_) * VectNorm(a_,
b_) - (
                        pow((

(pow((VectNorm(a_, d_) - VectNorm(b_, c_)), 2) +

VectNorm(a_, b_) * VectNorm(a_, b_) -
VectNorm(c_, d_) * VectNorm(c_, d_)) /
                                    (2 *
(VectNorm(a_, d_) - VectNorm(b_, c_)))
                                ), 2)
                )
        );
    }
}
template<class T>
trapeze<T>::trapeze(std::istream& is) {
    is >> a_ >> b_ >> c_ >> d_ ;
}
template<class T>
void trapeze<T>::print(std::ostream& os) const {
    os << "coordinate:\n"<<"\n"<< a_ << '\n' <<
b_ << '\n' << c_ << '\n' << d_ << '\n';
}
template<class T>
point<T> trapeze<T>::center() const {
    T x,y;
    x = (a_.x + b_.x + c_.x + d_.x ) / 4;
    y = (a_.y + b_.y + c_.y + d_.y) / 4;
    return {x,y};
}
#endif //LAB6_TRAPEZE_H
```

Main.cpp

```
#include <iostream>
#include "stack.h"
#include "trapeze.h"
#include <algorithm>
#include "my_allocator.h"
#include <map>
void menu() {
    std::cout << "1 - add(1 - push, 2 - insert  by iterator(enter index new
elem)\n"
                 "2 - delete(1 - pop, 2 - delete by iterator(enter
```

```
index)\n"
                 "3 - top\n"
                 "4 - print\n"
                 "5 - count if(enter max area)\n"
                 "6 - exit\n";
}
void usingStack() {
    int command, minicommand,index;
    double val;

container::Stack<trapeze<double>,my_all::my_allocator<trapeze<double>,330>>
st;
    for (;;) {
        std::cin >> command;
        if (command == 1) {
            try {
                std::cin >> minicommand;
                if (minicommand == 1) {
                    trapeze<double> p(std::cin);
                    st.Push(p);
                } else if (minicommand == 2) {
                    std::cin >> index;
                    try {
                        trapeze<double> p(std::cin);
                        st.Insert(index,p);
                    } catch (std::logic_error &e) {
                        std::cout << e.what() << std::endl;
                        continue;
                    }
                } }
            catch(std::bad_alloc& e) {
                std::cout << e.what() << std::endl;
                std::cout << "memory limit\n";
                continue; }
        } else if (command == 6) {
            break;
        } else if (command == 2) {
            std::cin >> minicommand;
            if (minicommand == 1) {
                try {
                    st.Pop();
                } catch (std::logic_error &e) {
                    std::cout << e.what() << std::endl;
                    continue;
                }
            }
            if (minicommand == 2) {
                std::cin >> index;
```

```
                try {
                    if (index < 0 || index > st.Size) {
                        throw std::logic_error("Out of bounds\n");
                    }
                    auto it = st.begin();
                    for (int i = 0; i < index; ++i) {
                        ++it;
                    }
                    st.Erase(it);
                }
                catch (std::logic_error &e) {
                    std::cout << e.what() << std::endl;
                    continue;
                }
            }
        } else if (command == 3) {
            try {
                st.Top().print(std::cout);
            }
            catch (std::logic_error &e) {
                std::cout << e.what() << std::endl;
                continue;
            }
        } else if (command == 4) {
            for (auto elem: st) {
                elem.print(std::cout);
            }
        } else if (command == 5) {
            std::cin >> val;
            std::cout << std::count_if(st.begin(), st.end(),
[val](trapeze<double> r) { return r.area() < val; })
                      << std::endl;
        } else {
            std::cout << "Error command\n";
            continue;
        }
    }
}
int main() {
    menu();
    usingStack();
    std::map<int, int, std::less<int>,
            my_all::my_allocator<std::pair<const int, int>, 80>> mp;
    for(int i = 0; i < 2; ++i){
        mp[i] = i;
    }
    for(int i = 2; i < 10; ++i){
        mp.erase(i - 2);
```

```
            mp[i] = i + 3;
        }
        return 0;
    }
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.14)
project(lab6)
set(CMAKE_CXX_STANDARD 17)
add_executable(lab6 main.cpp point.h trapeze.h queue.h stack.h
my_allocator.h)
```

2. Ссылка на репозиторий на GitHub

https://github.com/VSGolubev-bald/oop_exercise_06

3. Вывод :

Выполнив данную лабораторную работу, я впервые поработал с собственным аллокатором, на основе очереди.

Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Проектирование структуры классов
Цель работы :
Получение практических навыков в хороших практиках проектирования структуры классов

Студент:	Голубев В.С.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	4
Оценка:	
Дата:	

Москва, 2020

1. Код программы на языке C++:

Command.h

```
#ifndef OOP7_COMMAND_H
                        #define OOP7_COMMAND_H
                        #include "document.h"
                        struct Acommand {
                            virtual ~Acommand() = default;
                            virtual void UnExecute() = 0;
                        protected:
                            std::shared_ptr<document> doc_;
                        };
                        struct InsertCommand : public Acommand {
                        public:
                            void UnExecute() override;
                            InsertCommand(std::shared_ptr<document>&
                        doc);
                        };
                        struct DeleteCommand : public Acommand {
                        public:
                            DeleteCommand(std::shared_ptr<figure>&
                        newFigure, uint32_t
                        newIndex,std::shared_ptr<document>& doc);
                            void UnExecute() override;
                        private:
                            std::shared_ptr<figure> figure_;
                            uint32_t index_;
                        };
                        #endif //OOP7_COMMAND_H
```

Command.cpp

```
#include
"command.h"
                    void InsertCommand::UnExecute() {
                        doc_->RemoveLast();
                    }
                    InsertCommand::InsertCommand(std::shared_ptr<document>
                    &doc) {
                        doc_ = doc;
                    }
                    DeleteCommand::DeleteCommand(std::shared_ptr<figure>
                    &newFigure, uint32_t newIndex,
                    std::shared_ptr<document> &doc) {
                        doc_ = doc;
```

```
    figure_ = newFigure;
    index_ = newIndex;
}
void DeleteCommand::UnExecute() {
    doc_->InsertIndex(figure_,index_);
}
```

Document.h

```
#ifndef OOP7_DOCUMENT_H
#define OOP7_DOCUMENT_H
#include <fstream>
#include <cstdint>
#include <memory>
#include <string>
#include <algorithm>
#include "figure.h"
#include <vector>
#include "factory.h"
struct document {
public:
    void Print() const ;
    document(std::string& newName):
name_(newName), factory_(), buffer_(0) {};
    void Insert(std::shared_ptr<figure>& ptr);
    void Rename(const std::string& newName);
    void Save (const std::string& filename)
const;
    void Load(const std::string& filename);
    std::shared_ptr<figure> GetFigure(uint32_t
index);
    void Erase(uint32_t index);
    std::string GetName();
    size_t Size();
private:
    friend class InsertCommand;
    friend class DeleteCommand;
    factory factory_;
    std::string name_;
    std::vector<std::shared_ptr<figure>>
buffer_;
    void RemoveLast();
    void InsertIndex(std::shared_ptr<figure>&
newFigure, uint32_t index);
};
#endif //OOP7_DOCUMENT_H
```

Document.cpp

```
#include
"document.h"
                        void document::Print() const {
                            {
                                if (buffer_.empty()) {
                                    std::cout << "Buffer is empty\n";
                                }
                                for (auto elem : buffer_) {
                                    elem->print(std::cout);
                                }
                            }
                        }
                        void document::Insert(std::shared_ptr<figure>
                        &ptr) {
                            buffer_.push_back(ptr);
                        }
                        void document::Rename(const std::string &newName)
                        {
                            name_ = newName;
                        }
                        void document::Save(const std::string &filename)
                        const {
                            std::ofstream fout;
                            fout.open(filename);
                            if (!fout.is_open()) {
                                throw std::runtime_error("File is not
                        opened\n");
                            }
                            fout << buffer_.size() << '\n';
                            for (auto elem : buffer_) {
                                elem->printFile(fout);
                            }
                        }
                        void document::Load(const std::string &filename) {
                            std::ifstream fin;
                            fin.open(filename);
                            if (!fin.is_open()) {
                                throw std::runtime_error("File is not
                        opened\n");
                            }
                            size_t size;
                            fin >> size;
                            buffer_.clear();
                            for (int i = 0; i < size; ++i) {

                        buffer_.push_back(factory_.FigureCreateFile(fin));
```

```
    }
    name_ = filename;
}
std::shared_ptr<figure>
document::GetFigure(uint32_t index) {
    return buffer_[index];
}
void document::Erase(uint32_t index) {
    if ( index >= buffer_.size()) {
        throw std::logic_error("Out of bounds\n");
    }
    buffer_[index] = nullptr;
    for (; index < buffer_.size() - 1; ++index) {
        buffer_[index] = buffer_[index + 1];
    }
    buffer_.pop_back();
}
std::string document::GetName() {
    return this->name_;
}
size_t document::Size() {
    return buffer_.size();
}
void document::RemoveLast() {
    if (buffer_.empty()) {
        throw std::logic_error("Document is
empty");
    }
    buffer_.pop_back();
}
void document::InsertIndex(std::shared_ptr<figure>
&newFigure, uint32_t index) {
    buffer_.insert(buffer_.begin() + index,
newFigure);
}
```

Editor.h

```
#ifndef OOP7_EDITOR_H
#define OOP7_EDITOR_H
#include "figure.h"
#include "document.h"
#include <stack>
#include "command.h"
struct editor {
private:
    std::shared_ptr<document> doc_;
    std::stack<std::shared_ptr<Acommand>>
history_;
```

```
public:
    ~editor() = default;
    void PrintDocument();
    void CreateDocument(std::string& newName);
    bool DocumentExist();
    editor() : doc_(nullptr), history_()
    {
    }
    void
InsertInDocument(std::shared_ptr<figure>&
newFigure);
    void DeleteInDocument(uint32_t index);
    void SaveDocument();
    void LoadDocument(std::string& name);
    void Undo();
};
#endif //OOP7_EDITOR_H
```

Editor.cpp

```
#include "editor.h"
void editor::PrintDocument() {
    if (doc_ == nullptr) {
        std::cout << "No document!\n";
        return;
    }
    doc_->Print();
}
void editor::CreateDocument(std::string &newName) {
    doc_ = std::make_shared<document>(newName);
}
bool editor::DocumentExist() {
    return doc_ != nullptr;
}
void
editor::InsertInDocument(std::shared_ptr<figure>
&newFigure) {
    if (doc_ == nullptr) {
        std::cout << "No document!\n";
        return;
    }
    std::shared_ptr<Acommand> command =
std::shared_ptr<Acommand>(new InsertCommand(doc_));
    doc_->Insert(newFigure);
    history_.push(command);
}
void editor::DeleteInDocument(uint32_t index) {
```

```
    if (doc_ == nullptr) {
        std::cout << "No document!\n";
        return;
    }
    if (index >= doc_->Size()) {
        std::cout << "Out of bounds\n";
        return;
    }
    std::shared_ptr<figure> tmp = doc_-
>GetFigure(index);
    std::shared_ptr<Acommand> command =
std::shared_ptr<Acommand>(new
DeleteCommand(tmp,index,doc_));
    doc_->Erase(index);
    history_.push(command);
}
void editor::SaveDocument() {
    if (doc_ == nullptr) {
        std::cout << "No document!\nNot ";
        return;
    }
    std::string saveName = doc_->GetName();
    doc_ ->Save(saveName);
}
void editor::LoadDocument(std::string &name) {
    try {
        doc_ = std::make_shared<document>(name);
        doc_->Load(name);
        while (!history_.empty()){
            history_.pop();
        }
    } catch(std::logic_error& e) {
        std::cout << e.what();
    }
}
void editor::Undo() {
    if (history_.empty()) {
        throw std::logic_error("History is
empty\n");
    }
    std::shared_ptr<Acommand> lastCommand =
history_.top();
    lastCommand->UnExecute();
    history_.pop();
}
```

Factory.h

```
#ifndef OOP7_FACTORY_H
                        #define OOP7_FACTORY_H
                        #include <memory>
                        #include <iostream>
                        #include <fstream>
                        #include "trapeze.h"
                        #include "rhombus.h"
                        #include "pentagon.h"
                        #include <string>
                        struct factory {
                            std::shared_ptr<figure>
                        FigureCreate(std::istream& is);
                            std::shared_ptr<figure>
                        FigureCreateFile(std::ifstream& is);
                        };
                        #endif //OOP7_FACTORY_H
```

Factory.cpp

```
#include "factory.h"
                      std::shared_ptr<figure>
                      factory::FigureCreate(std::istream &is) {
                          std::string name;
                          is >> name;
                          if ( name == "pentagon" ) {
                              return std::shared_ptr<figure> ( new
                      pentagon(is));
                          } else if ( name == "trapeze") {
                              return std::shared_ptr<figure> ( new
                      trapeze(is));
                          } else if ( name == "rhombus") {
                              return std::shared_ptr<figure> ( new
                      rhombus(is));
                          } else {
                              throw std::logic_error("There is no such
                      figure\n");
                          }
                      }
                      std::shared_ptr<figure>
                      factory::FigureCreateFile(std::ifstream &is) {
                          std::string name;
                          is >> name;
                          if ( name == "pentagon" ) {
                              return std::shared_ptr<figure> ( new
                      pentagon(is));
                          } else if ( name == "rhombus") {
```

```
        return std::shared_ptr<figure> ( new
trapeze(is));
    } else if ( name == "rhombus") {
        return std::shared_ptr<figure> ( new
rhombus(is));
    } else {
        throw std::logic_error("There is no such
figure\n");
    }
}
```

Figure.h

```
#ifndef OOP7_FIGURE_H
#define OOP7_FIGURE_H
#include <iostream>
#include "point.h"
#include <fstream>
struct figure {
    virtual point center() const = 0;
    virtual void print(std::ostream&) const = 0 ;
    virtual void printFile(std::ofstream&) const
= 0 ;
    virtual double square() const = 0;
    virtual ~figure() = default;
};
#endif //OOP7_FIGURE_H
```

Pentagon.h

```
#ifndef
OOP7_PENTAGON_H
#define OOP7_PENTAGON_H
#include "figure.h"
struct pentagon : figure{
private:
    point a_,b_,c_,d_,e_;
public:
    point center() const override ;
    void print(std::ostream&) const  override ;
    void printFile(std::ofstream&) const  override
;
    double square() const override ;
    pentagon() = default;
    pentagon(std::istream& is);
    pentagon(std::ifstream& is);
};
```

Pentagon.cpp

```
#include
"pentagon.h"
#include "point.h"
point pentagon::center() const {
    double x,y;
    x = (a_.x + b_.x + c_.x + d_.x
+ e_.x) / 5;
    y = (a_.y + b_.y + c_.y + d_.y
+ e_.y) / 5;
    point p(x,y);
    return p;
}
void pentagon::print(std::ostream&
os) const {
    os << "pentagon\n"<< a_ <<
'\n' << b_ << '\n' << c_ << '\n'
<< d_ << '\n' << e_ << '\n';
}
void
pentagon::printFile(std::ofstream&
of) const {
    of << "pentagon\n"<< a_ <<
'\n' << b_ << '\n' << c_ << '\n'
<< d_ << '\n' << e_ << '\n';
}
double pentagon::square() const {
    return TrAngle(a_, b_, c_) +
TrAngle(c_, d_, e_) + TrAngle(a_,
c_, e_);
}
pentagon::pentagon(std::istream&
is) {
    is >> a_ >> b_ >> c_ >> d_ >>
e_;
}
pentagon::pentagon(std::ifstream&
is) {
    is >> a_ >> b_ >> c_ >> d_ >>
e_;
}
```

Point.h

```
#ifndef OOP7_POINT_H
                      #define OOP7_POINT_H
                      #include <iostream>
                      struct point {
                          double x, y;
                          point (double  a,double b) { x = a, y = b;};
                          point() = default;
                      };
                      std::istream& operator >> (std::istream& is,point&
                      p );
                      std::ostream& operator << (std::ostream& os,const
                      point& p);
                      point operator-(point l, point r);
                      double VectNorm(point l, point r);
                      double VectProd(point l, point r);
                      double ScalProd(point l, point r);
                      double TrAngle(point a, point b, point c);
                      #endif //OOP7_POINT_H
```

Point.cpp

```
#include "point.h"
                    #include <cmath>
                    std::istream& operator >> (std::istream& is,point& p
                    ) {
                        return  is >> p.x >> p.y;
                    }
                    std::ostream& operator << (std::ostream& os,const
                    point& p) {
                        return os << p.x <<' '<< p.y;
                    }
                    point operator-(point l, point r) {
                        return {r.x - l.x, r.y - l.y};
                    }
                    double VectNorm(point l, point r) {
                        point vect = operator-(l, r);
                        double res = sqrt(vect.x * vect.x + vect.y *
                    vect.y);
                        return res;
                    }
                    double TrAngle(point a, point b, point c) {
                        point  v1{}, v2{};
                        v1 = operator-(a, b);
                        v2 = operator-(a, c);
                        return std::abs(v1.x * v2.y - v2.x * v1.y) / 2;
                    }
                    double VectProd(point l, point r) {
```

```
    return l.x * r.y - r.x * l.y;
}
double ScalProd(point l, point r) {
    return std::abs(l.x * r.x + l.y * r.y);
}
```

Rhombus.h

```
#ifndef OOP7_RHOMBUS_H
#define OOP7_RHOMBUS_H
#include "figure.h"
struct rhombus : figure{
private:
    point a_,b_,c_,d_;
public:
    point center() const override ;
    void print(std::ostream&) const  override ;
    void printFile(std::ofstream&) const
override ;
    double square() const override ;
    rhombus() = default;
    rhombus(std::istream& is);
    rhombus(std::ifstream& is);
};
#endif //OOP7_RHOMBUS_H
```

Rhombus.cpp

```
#include "rhombus.h"
point rhombus::center() const {
    double x,y;
    x = (a_.x + b_.x + c_.x + d_.x) / 4;
    y = (a_.y + b_.y + c_.y + d_.y) / 4;
    point p(x,y);
    return p;
}
void rhombus::print(std::ostream& os) const {
    os << "rhombus\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << '\n';
}
void rhombus::printFile(std::ofstream& of) const {
    of << "rhombus\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << '\n';
}
double rhombus::square() const {
    return VectNorm(c_, a_) * VectNorm(d_, b_) /
2;
}
rhombus::rhombus(std::istream& is) {
```

```
    is >> a_ >> b_ >> c_ >> d_;
}
rhombus::rhombus(std::ifstream& is) {
    is >> a_ >> b_ >> c_ >> d_;
}
```

Trapeze.h

```
#ifndef OOP7_TRAPEZE_H
#define OOP7_TRAPEZE_H
#include "figure.h"
struct trapeze : figure{
private:
    point a_,b_,c_,d_;
public:
    point center() const override ;
    void print(std::ostream&) const  override ;
    void printFile(std::ofstream&) const
override ;
    double square() const override ;
    trapeze() = default;
    trapeze(std::istream& is);
    trapeze(std::ifstream& is);
};
#endif //OOP7_TRAPEZE_H
```

Trapeze.cpp

```
#include "trapeze.h"
#include <cmath>
point trapeze::center() const {
    double x,y;
    x = (a_.x + b_.x + c_.x + d_.x) / 4;
    y = (a_.y + b_.y + c_.y + d_.y) / 4;
    point p(x,y);
    return p;
}
void trapeze::print(std::ostream& os) const {
    os << "trapeze\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << "\n";
}
void trapeze::printFile(std::ofstream &of) const {
    of << "trapeze\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << "\n";
}
double trapeze::square() const {
        if ( (VectProd(operator-(a_,b_), operator-
(c_,d_)) == 0) && (VectProd(operator-(b_,c_),
```

```
operator-(a_,d_)) == 0) ) {
            return std::fabs((VectProd(operator-
(a_,b_), operator-(a_,d_)))) ;
        } else if (VectProd(operator-(a_,b_),
operator-(d_,c_)) == 0) {
            return ((VectNorm(a_, b_) +
VectNorm(d_, c_)) / 2) * sqrt(
                    VectNorm(d_, a_) *
VectNorm(d_, a_) - (
                            pow((

(pow((VectNorm(d_, c_) - VectNorm(a_, b_)), 2) +

VectNorm(d_, a_) * VectNorm(d_, a_) - VectNorm(b_,
c_) * VectNorm(b_, c_)) /
                                        (2 *
(VectNorm(d_, c_) - VectNorm(a_, b_)))
                                ), 2)
                    )
            );
        } else if (VectProd(operator-(b_,c_),
operator-(a_,d_)) == 0) {
            return ((VectNorm(b_, c_) +
VectNorm(a_, d_)) / 2) * sqrt(
                    VectNorm(a_, b_) *
VectNorm(a_, b_) - (
                            pow((

(pow((VectNorm(a_, d_) - VectNorm(b_, c_)), 2) +

VectNorm(a_, b_) * VectNorm(a_, b_) - VectNorm(c_,
d_) * VectNorm(c_, d_)) /
                                        (2 *
(VectNorm(a_, d_) - VectNorm(b_, c_)))
                                ), 2)
                    )
            );
        }
    }
trapeze::trapeze(std::istream& is) {
    is >> a_ >> b_ >> c_ >> d_;
}
trapeze::trapeze(std::ifstream& is) {
    is >> a_ >> b_ >> c_ >> d_;
}
```

Main.cpp

```
#include <iostream>
                    #include "factory.h"
                    #include "editor.h"
                    void help() {
                        std::cout << "create\n"
                                     "load\n"
                                     "save\n"
                                     "add\n"
                                     "remove\n"
                                     "print\n"
                                     "undo\n"
                                     "exit\n";
                    }
                    void create(editor& edit) {
                        std::string tmp;
                        std::cout << "Enter name of new document\n";
                        std::cin >> tmp;
                        edit.CreateDocument(tmp);
                        std::cout << "Document create\n";
                    }
                    void load(editor& edit) {
                        std::string tmp;
                        std::cout << "Enter path to the file\n";
                        std::cin >> tmp;
                        try {
                            edit.LoadDocument(tmp);
                            std::cout << "Document loaded\n";
                        } catch (std::runtime_error& e) {
                            std::cout << e.what();
                        }
                    }
                    void save(editor& edit) {
                        std::string tmp;
                        try {
                            edit.SaveDocument();
                            std::cout << "save document\n";
                        } catch (std::runtime_error& e) {
                            std::cout << e.what();
                        }
                    }
                    void add(editor& edit) {
                        factory fac;
                        try {
                            std::shared_ptr<figure> newElem =
                    fac.FigureCreate(std::cin);
```

```
            edit.InsertInDocument(newElem);
        } catch (std::logic_error& e) {
            std::cout << e.what() << '\n';
        }
        std::cout << "Ok\n";
    }
    void remove(editor& edit) {
        uint32_t index;
        std::cout << "Enter index\n";
        std::cin >> index;
        try {
            edit.DeleteInDocument(index);
            std::cout << "Ok\n";
        } catch (std::logic_error& err) {
            std::cout << err.what() << "\n";
        }
    }
    int main() {
        editor edit;
        std::string command;
        std::cout << "Commands:" << std::endl;
        help();
        while (true) {
            std::cin >> command;
            if (command == "create") {
                create(edit);
            } else if (command == "load") {
                load(edit);
            } else if (command == "save") {
                save(edit);
            } else if (command == "exit") {
                break;
            } else if (command == "add") {
                add(edit);
            } else if (command == "remove") {
                remove(edit);
            } else if (command == "print") {
                edit.PrintDocument();
            } else if (command == "undo") {
                try {
                    edit.Undo();
                } catch (std::logic_error& e) {
                    std::cout << e.what();
                }
            } else {
                std::cout << "Unknown command\n";
            }
        }
```

```
    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10.2)
project(oop_exercise_07)
set(CMAKE_CXX_STANDARD 17)
add_executable(oop_exercise_07
main.cpp point.h trapeze.h
trapeze.cpp figure.h point.cpp
pentagon.cpp pentagon.h
rhombus.cpp rhombus.h document.h
factory.h command.h editor.h
document.cpp factory.cpp
editor.cpp command.cpp)
```

2. Ссылка на репозиторий на GitHub

https://github.com/VSGolubev-bald/oop_exercise_07

3. Вывод :

Выполнив данную лабораторную работу, я обучился азам работы с структурами классов

Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Асинхронное программирование
Цель работы :
Знакомство с асинхронным программированием;
Получение навыков в параллельной обработке данных;
Получение навыков в синхронизации потоков

Студент:	Голубев В.С.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	4
Оценка:	
Дата:	

Москва, 2020

1. Код программы на языке C++:

Factory.h

```
#ifndef LAB8_FACTORY_H
                        #define LAB8_FACTORY_H
                        #include <memory>
                        #include <iostream>
                        #include <fstream>
                        #include "trapeze.h"
                        #include "rhombus.h"
                        #include "pentagon.h"
                        #include <string>
                        struct factory {
                            std::shared_ptr<figure>
                        FigureCreate(std::istream& is);
                            std::shared_ptr<figure>
                        FigureCreateFile(std::ifstream& is);
                        };
                        #endif //LAB8_FACTORY_H
```

Factory.cpp

```
#include "factory.h"
                        std::shared_ptr<figure>
                        factory::FigureCreate(std::istream &is) {
                            std::string name;
                            is >> name;
                            if ( name == "pentagon" ) {
                                return std::shared_ptr<figure> ( new
                        pentagon(is));
                            } else if ( name == "trapeze") {
                                return std::shared_ptr<figure> ( new
                        trapeze(is));
                            } else if ( name == "rhombus") {
                                return std::shared_ptr<figure> ( new
                        rhombus(is));
                            } else {
                                throw std::logic_error("There is no such
                        figure\n");
                            }
                        }
                        std::shared_ptr<figure>
                        factory::FigureCreateFile(std::ifstream &is) {
                            std::string name;
                            is >> name;
                            if ( name == "pentagon" ) {
```

```
            return std::shared_ptr<figure> ( new
        pentagon(is));
    } else if ( name == "rhombus") {
            return std::shared_ptr<figure> ( new
        trapeze(is));
    } else if ( name == "rhombus") {
            return std::shared_ptr<figure> ( new
        rhombus(is));
    } else {
            throw std::logic_error("There is no such
        figure\n");
    }
}
```

Figure.h

```
#ifndef LAB8_FIGURE_H
#define LAB8_FIGURE_H
#include <iostream>
#include "point.h"
#include <fstream>
struct figure {
    virtual point center() const = 0;
    virtual void print(std::ostream&) const = 0 ;
    virtual void printFile(std::ofstream&) const
= 0 ;
    virtual double square() const = 0;
    virtual ~figure() = default;
};
#endif //LAB8_FIGURE_H
```

Pentagon.h

```
#ifndef LAB8_PENTAGON_H
#define LAB8_PENTAGON_H
#include "figure.h"
struct pentagon : figure{
private:
    point a_,b_,c_,d_,e_;
public:
    point center() const override ;
    void print(std::ostream&) const  override ;
    void printFile(std::ofstream&) const
override ;
    double square() const override ;
```

```
        pentagon() = default;
        pentagon(std::istream& is);
        pentagon(std::ifstream& is);
    };
    #endif //LAB8_PENTAGON_H
```

Pentagon.cpp

```
#include "pentagon.h"
#include "point.h"
point pentagon::center() const {
    double x,y;
    x = (a_.x + b_.x + c_.x + d_.x + e_.x) / 5;
    y = (a_.y + b_.y + c_.y + d_.y + e_.y) / 5;
    point p(x,y);
    return p;
}
void pentagon::print(std::ostream& os) const {
    os << "pentagon\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << '\n' << e_ << '\n';
}
void pentagon::printFile(std::ofstream& of) const
{
    of << "pentagon\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << '\n' << e_ << '\n';
}
double pentagon::square() const {
    return TrAngle(a_, b_, c_) + TrAngle(c_, d_,
e_) + TrAngle(a_, c_, e_);
}
pentagon::pentagon(std::istream& is) {
    is >> a_ >> b_ >> c_ >> d_ >> e_;
}
pentagon::pentagon(std::ifstream& is) {
    is >> a_ >> b_ >> c_ >> d_ >> e_;
}
```

Point.h

```
#ifndef LAB8_POINT_H
#define LAB8_POINT_H
#include <iostream>
struct point {
    double x, y;
    point (double  a,double b) { x = a, y = b;};
```

```
        point() = default;
    };
    std::istream& operator >> (std::istream& is,point&
    p );
    std::ostream& operator << (std::ostream& os,const
    point& p);
    point operator-(point l, point r);
    double VectNorm(point l, point r);
    double VectProd(point l, point r);
    double ScalProd(point l, point r);
    double TrAngle(point a, point b, point c);
    #endif //LAB8_POINT_H
```

Point.cpp

```
#include "point.h"
#include <cmath>
std::istream& operator >> (std::istream& is,point& p
) {
    return  is >> p.x >> p.y;
}
std::ostream& operator << (std::ostream& os,const
point& p) {
    return os << p.x <<' '<< p.y;
}
point operator-(point l, point r) {
    return {r.x - l.x, r.y - l.y};
}
double VectNorm(point l, point r) {
    point vect = operator-(l, r);
    double res = sqrt(vect.x * vect.x + vect.y *
vect.y);
    return res;
}
double TrAngle(point a, point b, point c) {
    point  v1{}, v2{};
    v1 = operator-(a, b);
    v2 = operator-(a, c);
    return std::abs(v1.x * v2.y - v2.x * v1.y) / 2;
}
double VectProd(point l, point r) {
    return l.x * r.y - r.x * l.y;
}
double ScalProd(point l, point r) {
    return std::abs(l.x * r.x + l.y * r.y);
}
```

Processor.h

```
#ifndef
LAB8_PROCESSOR_H
                  #define LAB8_PROCESSOR_H
                  #include <iostream>
                  #include <condition_variable>
                  #include <thread>
                  #include <vector>
                  #include <mutex>
                  #include "factory.h"
                  #include "figure.h"
                  struct processor {
                      virtual void
                  process(std::shared_ptr<std::vector<std::shared_ptr<figure>>>
                  buffer) = 0;
                  };
                  struct stream_processor : processor {
                      void
                  process(std::shared_ptr<std::vector<std::shared_ptr<figure>>>
                  buffer) override;
                  };
                  struct file_processor : processor {
                      void
                  process(std::shared_ptr<std::vector<std::shared_ptr<figure>>>
                  buffer) override;
                  private:
                      uint64_t counter = 0;
                  };
                  #endif //LAB8_PROCESSOR_H
```

Processor.cpp

```
#include
"processor.h"
               void
               stream_processor::process(std::shared_ptr<std::vector<std::shared_p
               buffer) {
                   for (const auto& figure : *buffer) {
                       figure->print(std::cout);
                   }
               }
               void
               file_processor::process(std::shared_ptr<std::vector<std::shared_ptr
               buffer) {
                   std::ofstream fout;
                   fout.open(std::to_string(counter) + ".txt");
                   ++counter;
```

```
        if (!fout.is_open()) {
            std::cout << "File not opened\n";
            return;
        }
        for (const auto& figure : *buffer) {
            figure->printFile(fout);
        }
    }
```

Rhombus.h

```
#ifndef LAB8_RHOMBUS_H
#define LAB8_RHOMBUS_H
#include "figure.h"
struct rhombus : figure{
private:
    point a_,b_,c_,d_;
public:
    point center() const override ;
    void print(std::ostream&) const  override ;
    void printFile(std::ofstream&) const
override ;
    double square() const override ;
    rhombus() = default;
    rhombus(std::istream& is);
    rhombus(std::ifstream& is);
};
#endif //LAB8_RHOMBUS_H
```

Rhpmbus.cpp

```
#include "rhombus.h"
point rhombus::center() const {
    double x,y;
    x = (a_.x + b_.x + c_.x + d_.x) / 4;
    y = (a_.y + b_.y + c_.y + d_.y) / 4;
    point p(x,y);
    return p;
}
void rhombus::print(std::ostream& os) const {
    os << "rhmobus\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << '\n';
}
void rhombus::printFile(std::ofstream& of) const {
    of << "rhombus\n"<< a_ << '\n' << b_ << '\n'
```

```
<< c_ << '\n' << d_ << '\n';
}
double rhombus::square() const {
    return VectNorm(c_, a_) * VectNorm(d_, b_) /
2;
}
rhombus::rhombus(std::istream& is) {
    is >> a_ >> b_ >> c_ >> d_;
}
rhombus::rhombus(std::ifstream& is) {
    is >> a_ >> b_ >> c_ >> d_;
}
```

Subscriber.h

```
#ifndef
LAB8_SUBSCRIBER_H
#define LAB8_SUBSCRIBER_H
#include <iostream>
#include <condition_variable>
#include <thread>
#include <vector>
#include <mutex>
#include "factory.h"
#include "figure.h"
#include "processor.h"
struct subscriber {
    void operator()();
    std::vector<std::shared_ptr<processor>>
processors;

std::shared_ptr<std::vector<std::shared_ptr<figure>>>
buffer;
    std::mutex mtx;
    std::condition_variable cv;
    bool end = false;
};
#endif //LAB8_SUBSCRIBER_H
```

Subscriber.cpp

```
#include "subscriber.h"
 void subscriber::operator()() {
     while(true) {
         std::unique_lock<std::mutex> lock(mtx);
         cv.wait(lock,[&]{ return (buffer !=
     nullptr || end);});
```

```
                if (end) {
                    break;
                }
                for (const auto& processor_elem:
   processors) {
                    processor_elem->process(buffer);
                }
                buffer = nullptr;
                cv.notify_all();
            }
        }
```

Trapeze.h

```
#ifndef LAB8_TRAPEZE_H
#define LAB8_TRAPEZE_H
#include "figure.h"
struct trapeze : figure{
private:
    point a_,b_,c_,d_;
public:
    point center() const override ;
    void print(std::ostream&) const  override ;
    void printFile(std::ofstream&) const
override ;
    double square() const override ;
    trapeze() = default;
    trapeze(std::istream& is);
    trapeze(std::ifstream& is);
};
#endif //LAB8_TRAPEZE_H
```

Trapeze.cpp

```
#include "trapeze.h"
#include <cmath>
point trapeze::center() const {
    double x,y;
    x = (a_.x + b_.x + c_.x + d_.x) / 4;
    y = (a_.y + b_.y + c_.y + d_.y) / 4;
    point p(x,y);
    return p;
}
void trapeze::print(std::ostream& os) const {
    os << "trapeze\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << "\n";
}
```

```
void trapeze::printFile(std::ofstream &of) const {
    of << "trapeze\n"<< a_ << '\n' << b_ << '\n'
<< c_ << '\n' << d_ << "\n";
}
double trapeze::square() const {
    if ( (VectProd(operator-(a_,b_), operator-
(c_,d_)) == 0) && (VectProd(operator-(b_,c_),
operator-(a_,d_)) == 0) ) {
        return std::fabs((VectProd(operator-
(a_,b_), operator-(a_,d_)))) ;
    } else if (VectProd(operator-(a_,b_),
operator-(d_,c_)) == 0) {
        return ((VectNorm(a_, b_) + VectNorm(d_,
c_)) / 2) * sqrt(
                VectNorm(d_, a_) * VectNorm(d_,
a_) - (
                        pow((

(pow((VectNorm(d_, c_) - VectNorm(a_, b_)), 2) +
                                        VectNorm(d_,
a_) * VectNorm(d_, a_) - VectNorm(b_, c_) *
VectNorm(b_, c_)) /
                                    (2 *
(VectNorm(d_, c_) - VectNorm(a_, b_)))
                                ), 2)
                )
        );
    } else if (VectProd(operator-(b_,c_),
operator-(a_,d_)) == 0) {
        return ((VectNorm(b_, c_) + VectNorm(a_,
d_)) / 2) * sqrt(
                VectNorm(a_, b_) * VectNorm(a_,
b_) - (
                        pow((

(pow((VectNorm(a_, d_) - VectNorm(b_, c_)), 2) +
                                        VectNorm(a_,
b_) * VectNorm(a_, b_) - VectNorm(c_, d_) *
VectNorm(c_, d_)) /
                                    (2 *
(VectNorm(a_, d_) - VectNorm(b_, c_)))
                                ), 2)
                )
        );
    }
}
trapeze::trapeze(std::istream& is) {
    is >> a_ >> b_ >> c_ >> d_;
```

```
}
trapeze::trapeze(std::ifstream& is) {
    is >> a_ >> b_ >> c_ >> d_;
}
```

Main.cpp

```
#include
<iostream>
            #include <memory>
            #include "subscriber.h"
            int main(int argc,char* argv[]) {
                if (argc != 2) {
                    std::cout << " Usage:\n" << argv[0] << "    <buffer
            size>\n";
                    return 1;
                }
                const int32_t buffer_size = std::stoi(argv[1]);
                std::shared_ptr<std::vector<std::shared_ptr<figure>>>
            buffer =

            std::make_shared<std::vector<std::shared_ptr<figure>>>();
                buffer->reserve(buffer_size);
                factory factory;
                std::string command;
                std::cout << "add - adding a new shape\n" << "exit - the
            end of the program\n";
                //thread
                subscriber sub;

            sub.processors.push_back(std::make_shared<stream_processor>());

            sub.processors.push_back(std::make_shared<file_processor>());
                std::thread sub_thread(std::ref(sub));
                while (true) {
                    std::unique_lock<std::mutex> guard(sub.mtx);
                    std::cout << "begin\n";
                    std::cin >> command;
                    if (command == "add") {
                        try {
                            buffer-
            >push_back(factory.FigureCreate(std::cin));
                        } catch (std::logic_error &e) {
                            std::cout << e.what() << '\n';
                            continue;
                        }
```

```
                    if (buffer->size() == buffer_size) {
                        sub.buffer = buffer;
                        sub.cv.notify_all();
                        sub.cv.wait(guard, [&](){ return sub.buffer ==
        nullptr;});
                        buffer->clear();
                    }
                } else if (command == "exit")  {
                    break;
                } else {
                    std::cout << "unknown command\n";
                }
            }
            sub.end = true;
            sub.cv.notify_all();
            sub_thread.join();
            return 0;
        }
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.15)
                                        project(lab8)
                                        set(CMAKE_CXX_STANDARD 14)
                                        add_executable(lab8 main.cpp
                                        pentagon.cpp trapeze.cpp trapeze.h
                                        pentagon.cpp pentagon.h
                                        rhombus.cpp rhombus.h factory.cpp
                                        factory.h processor.cpp
                                        processor.h figure.h point.cpp
                                        point.h subscriber.cpp
                                        subscriber.h)
```

Makefile (пользовательский)

```
CC = g++
        FLAGS = -std=c++17 -Wall -pthread
        FILES = *.cpp
        PROG = run
        all:
            $(CC) $(FLAGS) -o $(PROG) $(FILES)
        clean:
            rm -f *.o run
```

2. Ссылка на репозиторий на GitHub

https://github.com/VSGolubev-bald/oop_exercise_05

3. Вывод :

Выполнив данную лабораторную работу, я получил опыт работы с потоками на C++