

Московский Авиационный Институт
(Национальный исследовательский Университет)

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

**Лабораторная работа
по курсу «ООП»**

**Тема:
Аллокатеры.**

Студент:	Черненко И.Д
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	24
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

octagon.h:

```
#ifndef OOP_EXERCISE_05_OCTAGON_H
#define OOP_EXERCISE_05_OCTAGON_H
#include <utility>
#include <cmath>
template<class T>
struct TOctagon{
    using type = T;
    using vertex = std::pair<T,T>;
    vertex A, B, C, D, E, F, G, H;
    TOctagon() :
        A(0,0), B(0,0), C(0,0), D(0,0), E(0,0), F(0,0), G(0,0), H(0,0)
    {}
    TOctagon(T x1, T y1, T x2, T y2, T x3, T y3, T x4, T y4, T x5, T y5, T x6, T y6, T
x7, T y7, T x8, T y8) :
        A(x1, y1), B(x2, y2), C(x3, y3), D(x4, y4), E(x5, y5), F(x6, y6), G(x7, y7),
H(x8, y8)
    {}
    TOctagon(vertex a, vertex b, vertex c, vertex d, vertex e, vertex f, vertex g, vertex
h) : A(a), B(b), C(c), D(d), E(e), F(f), G(g), H(h)
    {}
    std::pair<double,double> center() const;
    void print() const;
    double area() const;
};
```

```
template <class T>
```

```
double TOctagon<T>::area() const{
    return fabs((((A.first * B.second) + (B.first * C.second) + (C.first * D.second) +
(D.first * E.second) + (E.first * F.second) + (F.first * G.second) + (G.first *
H.second) + (H.first * A.second) - (B.first * A.second) - (C.first * B.second) - (D.first
* C.second) - (E.first * D.second) - (F.first * E.second) - (G.first * F.second) - (H.first
* G.second) - (A.first * H.second)) * 0.5);
}
```

```
template <class T>
```

```
void TOctagon<T>::print() const{
    std::cout << A << " " << B << " " << C << " " << D << " " << E << " " << F << " "
<< G << " " << H << "\n";
}
```

```

template <class T>

std::pair<double, double> TOctagon<T>::center() const{
    return std::make_pair(static_cast<double>(A.first + B.first + C.first + D.first +
E.first + F.first + G.first + H.first) / 8,static_cast<double>(A.second + B.second +
C.second + D.second + E.second + F.second + G.second + H.second) / 8);
}
#endif //OOP_EXERCISE_05_OCTAGON_H

```

vertex.h:

```

#ifndef VERTEX_H
#define VERTEX_H
#include <iostream>
template <typename T1, typename T2>
std::istream& operator>> (std::istream& is, std::pair<T1, T2>& p) {
    is >> p.first >> p.second;
    if (is.fail()) {
        throw std::logic_error("Wrong type");
    }
    return is;
}

template <typename T1, typename T2>
std::ostream& operator<< (std::ostream& out, const std::pair<T1, T2>& p) {
    out << "(" << p.first << ", " << p.second << ") ";
    return out;
}

template<class T>
std::pair<T,T> operator+(std::pair<T,T> lhs, std::pair<T,T> rhs){
    std::pair<T,T> res;
    res.first = lhs.first + rhs.first;
    res.second = lhs.second + rhs.second;
    return res;
}

template<class T>
std::pair<T, T> operator/=(std::pair<T,T> vertex, double val) {
    vertex.first = vertex.first / val;
    vertex.second = vertex.second / val;
    return vertex;
}
#endif //VERTEX_H

```

queue.h:

```
#ifndef QUEUE_H
#define QUEUE_H
#include <memory>
#include <exception>
#include <cstdint>
typedef unsigned long long ull;
namespace my_container {
    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

```

2. Ссылка на репозиторий на GitHub

https://github.com/IlICher/oop_exercise_06

3. Набор testcases.

test_00:

push

2

2

2

2

3

3

4

4

44

4

4

4

4

4

4

4

prt

test_01:

push

2

2

2

2

3

3

4

4

44

4

4

4

4

4

4

4

push

654

6456

456

456

456

457
567
567
56756
756
756
756
7567
567
567
567
add 0
4
4
4
44
4
4
4
4
44
4
4
4
4
4
4
4
4
prt
test_02:
4
4
4
4
4
4
4
4
push
654
6456
456
456
456
457

567
567
56756
756
756
756
7567
567
567
567
add 0
4
4
4
44
4
4
4
4
44
4
4
4
4
4
4
4
4
top
rmv 1
rmv 0
prt
pop
test_03:
push
0
0
00
0
0
0
0
0
0
0
00
0

0
0
0
0
0
push
1
1
2
2
3
3
4
4
5
5
6
6
7
7
8
8
prt
add
1
3
3
5
1
5
-1
3
-3
-3
-3
-5
-1
-5
1
-3
3
prt
check 0
check 1
check 10

check 10000
test_04:
rmv -1

4. Результаты выполнения тестов.

test_00:
(2, 2) (2, 2) (3, 3) (4, 4) (44, 4) (4, 4) (4, 4) (4, 4)
test_01:
(4, 4) (4, 44) (4, 4) (4, 4) (44, 4) (4, 4) (4, 4) (4, 4)

(2, 2) (2, 2) (3, 3) (4, 4) (44, 4) (4, 4) (4, 4) (4, 4)

(654, 6456) (456, 456) (456, 457) (567, 567) (56756, 756) (756, 756) (7567, 567)
(567, 567)

test_02:
(654, 6456) (456, 456) (456, 457) (567, 567) (56756, 756) (756, 756) (7567, 567)
(567, 567)

empty

test_03:
no such an element

test_04:
no such a figure

5. Объяснение результатов работы программы.

- 1) При запуске программы вводится одна из 8 возможных команд в виде строки.
- 2) add – пользователь вводит координаты восьмиугольника, который добавляется в очередь по индексу.
- 3) rmv – пользователь вводит индекс фигуры в очереди, которая впоследствии удаляется.
- 4) prt – выводятся координаты восьмиугольников в очереди.
- 5) check – вывод на экран кол-ва объектов, у которых площадь меньше заданной.
- 6) top – вывод вершины очереди.
- 7) pop – удаляет элемент из начала очереди.
- 8) push – добавляет элемент в начало очереди.

6. Вывод.

Выполняя данную лабораторную, я получил опыт работы с аллокаторами и реализовал его, на основе очереди.