Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Итераторы и умные указатели.

Студент:	Косогоров В.В.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	10
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

square.h:

```
#ifndef D_SQUARE_H_
#define D_SQUARE_H_

#include <iostream>
#include <assert.h>
#include <math.h>

#include "vertex.hpp"

template<class T>
struct Square {
    Square(std::istream &is);
    int IsCorrect() const;

    vertex<double> Center() const;
    void Print() const;
    double Area() const;

private:
    vertex<T> one,two,three,four;
};

template<class T>
Square<T>::Square(std::istream &is){
    is >> one >> two >> three >> four;
    assert(IsCorrect());
}

template<class T>
int Square<T>::IsCorrect() const {
    const T vec1_x = two.x - one.x;
    const T vec1_y = two.y - one.y;

    const T vec2_x = three.x - two.x;
    const T vec2_y = three.y - two.y;

    const T vec3_x = four.x - one.x;
    const T vec3_y = four.y - one.y;

    const T vec4_x = four.x - three.x;
    const T vec4_y = four.y - three.y;
```

```
    const T dotProduct1 = vec1_x * vec2_x + vec1_y * vec2_y;
    const T dotProduct2 = vec3_x * vec1_x + vec3_y * vec1_y;
    const T dotProduct3 = vec3_x * vec4_x + vec3_y * vec4_y;

    const T vec1_length = sqrt(vec1_x * vec1_x + vec1_y * vec1_y);
    const T vec2_length = sqrt(vec2_x * vec2_x + vec2_y * vec2_y);

    if (dotProduct1 == 0 && dotProduct2 == 0 && dotProduct3 == 0 && vec1_length
== vec2_length) {
        return 1;
    }
    return 0;
}

template<class T>
vertex<double> Square<T>::Center() const {
    vertex<double> center;
    center = (one + three) / 2;
    return center;
}

template<class T>
void Square<T>::Print() const {
    std::cout << "Vertices: " << one << " " << two << " " << three << " " << four << '\
n';
    std::cout << "Area:" << Area() << std::endl;
    std::cout << "Center:" << Center() << std::endl;
}

template<class T>
double Square<T>::Area() const {
    const T vecX = two.x - one.x;
    const T vecY = two.y - one.y;

    return vecX * vecX + vecY * vecY;
}

#endif
```

vertex.h:

```
#ifndef D_VERTEX_H_
#define D_VERTEX_H_
```

```
#include <iostream>

template<class T>
struct vertex {
   T x, y;
};

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

template<class T>
std::ostream& operator<< (std::ostream& os, const vertex<T>& v){
   os << "(" << v.x << ", " << v.y << ") ";
   return os;
}

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

template<class T>
vertex<T> operator/ (const vertex<T> vert, const int num) {
   vertex<T> res;
   res.x = vert.x / num;
   res.y = vert.y / num;
   return res;
}

#endif
```

list.h:

```
#ifndef _D_LIST_H
#define _D_LIST_H

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

```
namespace container {

template<class T>
class list {
private:
    struct node_t;

public:
    struct forward_iterator {
        using value_type = T;
        using reference = T&;
        using pointer = T*;
        using difference_type = ptrdiff_t;
        using iterator_category = std::forward_iterator_tag;

        forward_iterator(node_t* ptr) : ptr_(ptr) {};
        T& operator*();
        forward_iterator& operator++();
        forward_iterator operator++(int);
        bool operator==(const forward_iterator& it) const;
        bool operator!=(const forward_iterator& it) const;
        private:
            node_t* ptr_;
            friend list;
    };

    forward_iterator begin();
    forward_iterator end();
    void push(const T& value);
    void insert(const forward_iterator& it, const T& value);
    void insert(const int& pos, const T& value);
    void erase(const forward_iterator& it);
    void erase(int pos);
    void popFront();
    void printTail();
    list() = default;
    list(const list&) = delete;
    T operator[](int pos);

private:
    struct node_t {
        T value;
        std::unique_ptr<node_t> nextNode = nullptr;
        forward_iterator next();
```

```
        node_t(const T& value, std::unique_ptr<node_t> next) : value(value), nextNode(std::move(next)) {};
    };
    std::unique_ptr<node_t> head = nullptr;
    node_t* tail = nullptr;
    list& operator=(const list&);
};

template<class T>
typename list<T>::forward_iterator list<T>::node_t::next() {
    return nextNode.get();
}

template<class T>
T& list<T>::forward_iterator::operator*() {
    return ptr_->value;
}

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

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

template<class T>
bool list<T>::forward_iterator::operator!=(const forward_iterator& it) const {
    return ptr_ != it.ptr_;
}

template<class T>
bool list<T>::forward_iterator::operator==(const forward_iterator& it) const {
    return ptr_ == it.ptr_;
}

template<class T>
typename list<T>::forward_iterator list<T>::begin() {
    return head.get();
}
```

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

template<class T>
void list<T>::push(const T& value) {
    insert(this->begin(), value);
}

template<class T>
void list<T>::insert(const forward_iterator& it, const T& value) {
    std::unique_ptr<node_t> newNode(new node_t(value, nullptr));
    if (head == nullptr) {
        head = std::move(newNode);
    } else if (head->nextNode == nullptr) {
        if (it.ptr_) {
            tail = head.get();
            newNode->nextNode = std::move(head);
            head = std::move(newNode);
        } else {
            tail = newNode.get();
            head->nextNode = std::move(newNode);
        }
    } else if (head.get() == it.ptr_) {
        newNode->nextNode = std::move(head);
        head = std::move(newNode);
    } else if (it.ptr_ == nullptr) {
        tail->nextNode = std::move(newNode);
        tail = newNode.get();
    } else {
        auto temp = this->begin();
        while (temp.ptr_->next() != it.ptr_) {
            ++temp;
        }

        newNode->nextNode = std::move(temp.ptr_->nextNode);
        temp.ptr_->nextNode = std::move(newNode);
    }
}

template<class T>
void list<T>::insert(const int& pos, const T& value) {
    int i = 0;
```

```
    auto temp = this->begin();
    if (pos == 0) {
        insert(temp, value);
        return;
    }
    while(i < pos) {
        if (temp.ptr_ == nullptr) {
            break;
        }
        ++temp;
        ++i;
    }
    if (i < pos) {
        throw std::logic_error("Out of bounds");
    }
    this->insert(temp, value);
}

template<class T>
void list<T>::popFront() {
    if (list<T>::head == nullptr) {
        throw std::logic_error("no elements");
    }
    erase(list<T>::begin());
}

template<class T>
void list<T>::erase(const forward_iterator& it) {
    if (it == nullptr) {
        throw std::logic_error("Invalid iterator");
    }
    if(head == nullptr) {
        throw std::logic_error("Deleting from empty list");
    }
    if (it == this->begin()) {
        head = std::move(head->nextNode);
    } else {
        auto temp = this->begin();
        while(temp.ptr_->next() != it.ptr_) {
            ++temp;
        }
        temp.ptr_->nextNode = std::move(it.ptr_->nextNode);
    }
}
```

```
template<class T>
void list<T>::erase(int pos) {
    auto temp = this->begin();
    int i = 0;
    while (i < pos) {
        if(temp.ptr_ == nullptr) {
            break;
        }
        ++temp;
        ++i;
    }
    if (temp.ptr_ == nullptr) {
        throw std::logic_error("Out of bounds");
    }
    erase(temp);
}

template<class T>
T list<T>::operator[](int pos) {
    auto temp = this->begin();
    int i = 0;
    while (i < pos) {
        if (temp.ptr_ == nullptr) {
            break;
        }
        ++temp;
        ++i;
    }
    if (temp.ptr_ == nullptr) {
        throw std::logic_error("Out of bounds");
    } else {
        return temp.ptr_->value;
    }
}

}

#endif
```

main.cpp:

```
#include <iostream>
#include <algorithm>

#include "list.hpp"
```

```
#include "square.hpp"

int main() {
    container::list<Square<double>> list;
    int command, pos;

    while(true) {
        std::cout << std::endl;
        std::cout << "0 - quit" << std::endl;
        std::cout << "1 - add element to list by index or push front" << std::endl;
        std::cout << "2 - delete element from list (pop front / erase by index / erase by
iterator)" << std::endl;
        std::cout << "3 - range-based for print" << std::endl;
        std::cout << "4 - count_if example" << std::endl;
        std::cout << "5 - print element by [index]" << std::endl;
        std::cin >> command;

        if(command == 0) {
            break;

        } else if(command == 1) {
            std::cout << "Enter coordinates" << std::endl;
            Square<double> square(std::cin);

            std::cout << "1 - PushFront" << std::endl;
            std::cout << "2 - insert by index" << std::endl;
            std::cin >> command;
            if(command == 1) {
                list.push(square);
                continue;
            } else if(command == 2) {
                std::cout << "Enter index" << std::endl;
                std::cin >> pos;
                list.insert(pos, square);
                continue;
            } else {
                std::cout << "Wrong command" << std::endl;
                std::cin >> command;
                continue;
            }

        } else if(command == 2) {
            std::cout << "1 - erase by index" << std::endl;
            std::cout << "2 - erase by iterator" << std::endl;
            std::cout << "3 - pop front" << std::endl;
```

```
            std::cin >> command;
            if(command == 1) {
                std::cout << "Enter index" << std::endl;
                std::cin >> pos;
                list.erase(pos);
                continue;
            } else if(command == 2) {
                std::cout << "Enter index" << std::endl;
                std::cin >> pos;
                auto temp = list.begin();
                for(int i = 0; i < pos; ++i) {
                    ++temp;
                }
                list.erase(temp);
                continue;

            } else if (command == 3) {
                try {
                    list.popFront();
                } catch(std::exception& e) {
                    std::cout << e.what() << std::endl;
                    continue;
                }
            }
            else {
                std::cout << "Wrong command" << std::endl;
                std::cin >> command;
                continue;
            }

        } else if(command == 3) {
            for(const auto& item : list) {
                item.Print();
                continue;
            }

        } else if(command == 4) {
            std::cout << "Enter required area" << std::endl;
            std::cin >> pos;
            std::cout << "Number of squares with area less than " << pos << " equals ";
            std::cout << std::count_if(list.begin(), list.end(), [pos](Square<double>
square) {return square.Area() < pos;}) << std::endl;
            continue;

        } else if (command == 5) {
```

```
            std::cout << "Enter index to print for" << std::endl;
            std::cin >> pos;
            try {
                list[pos].Print();
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
                continue;
            }
            continue;

        } else {
            std::cout << "Wrong command" << std::endl;
            continue;
        }
    }

    return 0;
}
```

2. Ссылка на репозиторий на GitHub.

https://github.com/vladiq/oop_exercise_05

3. Набор тестов.

test_01.test:

```
1
1 0
1 1
0 1
0 0
1
2
3
0
```

test_02.test:

```
1
2 0
2 2
0 2
0 0
2
0
1
10 0
```

10 10
0 10
0 0
4
1000
5
0
0

test_03.test:
1
1 0
1 1
0 1
0 0
4
0
0

4. Результаты выполнения тестов.

test_01.result:

```
 0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
Enter coordinates
1 - PushFront
2 - insert by iterator

0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
1 - erase by index
2 - erase by iterator
3 - pop front
0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
```

test_02.result:

```
0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
Enter coordinates
1 - PushFront
2 - insert by iterator
Enter index

0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
Enter coordinates
1 - PushFront
2 - insert by iterator
Wrong command

0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
Enter index to print for
Vertices: (2, 0)  (2, 2)  (0, 2)  (0, 0)
Area:4
Center:(1, 1)

0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
```

test_03.result:

```
0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
Enter coordinates
1 - PushFront
2 - insert by iterator
Wrong command

0 - quit
1 - add element to list by iterator or PushFront
2 - delete element from list (pop front / erase by index / erase by ite
rator)
3 - range-based for print
4 - count_if example
5 - print element by [index]
```

5. Объяснение результатов работы программы.

Площадь квадрата находится путем получения квадрата стороны, а геометрический центр — как точка пересечения его диагоналей. Все ошибки в списке обрабатываются try-catch, а если пользователь вводит некорректную фигуру, то вызывается assert().

6. Вывод.

Выполняя данную лабораторную, я получил опыт работы с итераторами и умными указателями в C++. Узнал о применении итераторов в STL. Понял, как важно использовать свое пространство имен при создании своего контейнера. Умение работать с умными указателями позволяет избежать утечек памяти.