Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Отчёт по Лабораторной работе №5
“Основы работы с коллекциями: итераторы“
по курсу “Объектно-Объективное Программирование“
III Семестр

Студент:	Катермин В.С.
Группа:	М8О-208Б-18
Преподаватель:	Журавлёв А.А.
Оценка:	
Дата:	**28.12.19**

1. **Тема**: Основы работы с коллекциями: итераторы.
2. **Код программы:**

vertex.h

```
#ifndef D_VERTEX_H_
#define D_VERTEX_H_ 1

#include <iostream>

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

};

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

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

#endif // D_VERTEX_H_
```

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 / D_LIST_H_
```

square.h

```
#ifndef D_SQUARE_H_
#define D_SQUARE_H_ 1

#include <algorithm>
#include <iostream>
#include <cmath>
#include <cassert>

#include "vertex.h"

template<class T>
struct square {
   vertex<T> vertices[4];

   square(std::istream& is);

   vertex<double> center() const;
   double area() const;
   void print(std::ostream& os) const;

};

template<class T>
square<T>::square(std::istream& is) {
   for(int i = 0; i < 4; ++i){
      is >> vertices[i];
   }
   assert(((vertices[1].x - vertices[0].x)*(vertices[3].x - vertices[0].x))+((vertices[1].y - vertices[0].y)*(vertices[3].y
- vertices[0].y)) == 0);
   assert(((vertices[2].x - vertices[1].x)*(vertices[2].x - vertices[3].x))+((vertices[2].y - vertices[1].y)*(vertices[2].y
- vertices[3].y)) == 0);
   assert(((vertices[3].x - vertices[2].x)*(vertices[1].x - vertices[2].x))+((vertices[3].y - vertices[2].y)*(vertices[1].y
- vertices[2].y)) == 0);
   assert((vertices[1].x - vertices[0].x) == (vertices[0].y - vertices[3].y));
   assert((vertices[2].x - vertices[1].x) == (vertices[1].y - vertices[0].y));
   assert((vertices[3].x - vertices[2].x) == (vertices[2].y - vertices[1].y));
}

template<class T>
vertex<double> square<T>::center() const {
   return {(vertices[0].x + vertices[1].x + vertices[2].x + vertices[3].x) * 0.25, (vertices[0].y + vertices[1].y +
vertices[2].y + vertices[3].y) * 0.25};
}

template<class T>
double square<T>::area() const {
   const T d1 = vertices[0].x - vertices[1].x;
   const T d2 = vertices[3].x - vertices[0].x;
   return abs(d1 * d1) + abs(d2 * d2);
}

template<class T>
void square<T>::print(std::ostream& os) const {
    os << "Square ";
   for(int i = 0; i < 4; ++i){
      os << "[" << vertices[i] << "]";
      if(i + 1 != 4){
      os << " ";
      }
   }
```

```
    os << '\n';
}

#endif // D_SQUARE_H_

#include <iostream>
#include <algorithm>

#include "list.h"
#include "square.h"

enum Commands{
    cmd_quit,
    cmd_add,
    cmd_rmv,
    cmd_prntall,
    cmd_count,
    cmd_print
};

enum Add{
    add_push,
    add_idx
};

enum Remove{
    rmv_idx,
    rmv_itr,
    rmv_pop
};

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 (push front / by index)" << std::endl;
        // std::cout << "2 - Delete element from list (pop front / erase by index / erase by iterator)" << std::endl;
        // std::cout << "3 - Print all elements" << std::endl;
        // std::cout << "4 - Count_if example (with areas)" << std::endl;
        // std::cout << "5 - Print element by [index]" << std::endl;
        std::cin >> command;

        if(command == cmd_quit) {
            break;

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

            // std::cout << "0 - PushFront" << std::endl;
            // std::cout << "1 - Insert by index" << std::endl;
            std::cin >> command;
            if(command == add_push) {
                list.push(square);
                continue;
            } else if(command == add_idx) {
                // std::cout << "Enter index" << std::endl;
                std::cin >> pos;
                list.insert(pos, square);
                continue;
            } else {
```

```
            std::cout << "Command incorrect" << std::endl;
            std::cin >> command;
            continue;
        }

    } else if(command == 2) {
        // std::cout << "0 - Erase by index" << std::endl;
        // std::cout << "1 - Erase by iterator" << std::endl;
        // std::cout << "2 - Pop front" << std::endl;
        std::cin >> command;
        if(command == rmv_idx) {
            // std::cout << "Enter index" << std::endl;
            std::cin >> pos;
            list.erase(pos);
            continue;
        } else if(command == rmv_itr) {
            // 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 == rmv_pop) {
            try {
                list.popFront();
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
                continue;
            }
        }
        else {
            std::cout << "Command incorrect" << std::endl;
            std::cin >> command;
            continue;
        }

    } else if(command == cmd_prntall) {
        for(const auto& item : list) {
            item.print(std::cout);
            std::cout << "Center: [" << item.center() << "]" << std::endl;
            std::cout << "Area: " << item.area() << std::endl;
            continue;
        }

    } else if(command == cmd_count) {
        // 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 == cmd_print) {
        // std::cout << "Enter index to print for" << std::endl;
        std::cin >> pos;
        try {
            list[pos].print(std::cout);
            std::cout << "Center: [" << list[pos].center() << "]" << std::endl;
            std::cout << "Area: " << list[pos].area() << std::endl;
        } catch(std::exception& e) {
            std::cout << e.what() << std::endl;
```

```
            continue;
        }
        continue;

    } else {
        std::cout << "Command incorrect" << std::endl;
        continue;
    }
  }

  return 0;
}
```

CMakeLists.txt

```
project(lab5)

set(CMAKE_CXX_STANDARD 17)

add_executable(lab5
 ./main.cpp)

set(CMAKE_CXX_FLAGS
 "${CMAKE_CXX_FLAGS} -Wall -Wextra")
```

3. **Ссылка на репозиторий:**
   https://github.com/GitGood2000/oop_exercise_05

4. **Набор testcases:**

test_00.test

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

test_00.result

```
Square [-1 1] [0 2] [1 1] [0 0]
Center: [0 1]
Area: 2
```

test_01.test

```
1
0 2 2 3 3 1 1 0
0
1
1 3 4 6 7 3 4 0
0
3
4
10
5
1
2
1
3
0
```

test_01.result

```
Square [1 3] [4 6] [7 3] [4 0]
Center: [4 3]
Area: 18
```

Square [0 2] [2 3] [3 1] [1 0]
Center: [1.5 1.5]
Area: 5
Number of squares with area less than 10 equals 1
Square [0 2] [2 3] [3 1] [1 0]
Center: [1.5 1.5]
Area: 5

5. **Результаты выполнения тестов:**

user@PSB133S01ZFH:~/3sem_projects/oop_exercise_05/tests$ bash test.sh ../build/lab5
Test test_00.test: SUCCESS
Test test_01.test: SUCCESS

6. **Объяснение результатов работы программы:**

Программа выполняет определённые действия по введённым командам:

A) 0 — выход из программы;
B) 1 — добавление квадрата в список (методом push (0) или по индексу (1));
C) 2 — удаление элемента из списка (методом pop (0), по индексу (1) или по итератору (2));
D) 3 — вывод всех элементов списка в терминал;
E) 4 — считывание количества фигур, площадь которых меньше, чем [число] (образец count_if);
F) 5 — вывод определённой фигуры по определённому индексу;

Все ошибки в списке обрабатываются try-catch. При вводе некорректной фигуры запускается assert().

7. **Вывод:** 1) Ознакомились с итераторами и умными указателями в C++ и усвоили навык работы с ними; 2) Написана программа, производящая операции с помощью умных указателей и работающая с итераторами.