МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №7

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Мерц Савелий Павлович, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Задание:

Спроектировать и разработать итератор для динамической структуры данных, разработанную для лабораторной работы №6. Итератор должен быть разработан в виде шаблона и должен позволять работать с любыми типами фигур, согласно варианту задания.

Итератор должен позволять использовать структуру данных в операторах типа for. Например:

```
for(auto i : stack) {
      std::cout << *i << std::endl;
}
```

Вариант №14:

- Фигура: Восьмиугольник (Octagon)
- Контейнер: Динамический массив (TVector)

Описание программы:

Исходный код разделен на 10 файлов:

- point.hpp – описание класса точки
- point.cpp – реализация класса точки
- figure.hpp – описание класса фигуры
- octagon.hpp – описание класса восьмиугольника (наследуется от фигуры)
- octagon.cpp – реализация класса восьмиугольника
- tvector.hpp – реализация класса динамического массива
- iterator.hpp – реализация итератора
- main.cpp – основная программа

Дневник отладки:

Проблем не возникало

Вывод:

При выполнении работы я на практике познакомился с итераторами. Они позволяют легко реализовать обход всех элементов структуры данных, позволяют использовать цикл range-based-for и для самописных структур.

Исходный код:

point.hpp:

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
  Point();
  Point(double x, double y);

  double getX() const;
  double getY() const;

  friend std::istream& operator>>(std::istream& is, Point& p);
  friend std::ostream& operator<<(std::ostream& os, const Point& p);
  const Point& operator=(const Point& other);
  bool operator!=(const Point& other) const;
  bool operator==(const Point& other) const;
private:
  double x_;
  double y_;
};

#endif
```

point.cpp:

```
#include "point.h"

#include <cmath>

Point::Point() : x_(0.0), y_(0.0) {}

Point::Point(double x, double y) : x_(x), y_(y) {}

double Point::getX() const{
	return x_;
}

double Point::getY() const{
	return y_;
}

std::istream& operator>>(std::istream& is, Point& p) {
	is >> p.x_ >> p.y_;
	return is;
}

std::ostream& operator<<(std::ostream& os, const Point& p) {
	os << "(" << p.x_ << ", " << p.y_ << ")";
	return os;
}
```

```
const Point& Point::operator=(const Point& other) {
	if (this == &other)
	return *this;
	x_ = other.x_;
	y_ = other.y_;
	return *this;
}

bool Point::operator!=(const Point& other) const{
	if (x_ == other.x_) return false;
	if (y_ == other.y_) return false;
	return true;
}
bool Point::operator==(const Point& other) const{
	if (x_ != other.x_) return false;
	if (y_ != other.y_) return false;
	return true;
}
```

figure.hpp:

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>

class Figure {
public:
    size_t VertexesNumber();
    double Area();
    ~Figure() {};
};

#endif
```

octagon.hpp:

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include <iostream>

#include "figure.h"
#include "point.h"

class Octagon : public Figure {
public:
    Octagon();
     Octagon(Point a, Point b, Point c, Point d, Point e, Point f, Point g, Point
h);

    friend std::istream& operator>>(std::istream& is,  Octagon& obj);
    friend std::ostream& operator<<(std::ostream& os, const Octagon& obj);

    size_t VertexesNumber();
    double Area() const;
    const Octagon& operator=(const Octagon& other);
    bool operator==(const Octagon& other) const;
    bool operator!=(const Octagon& other) const;
```

```
    ~Octagon();
private:
    Point a_, b_, c_, d_, e_, f_, g_, h_;
};

#endif
```

octagon.cpp:

```
#include "octagon.h"

#include <cmath>

Octagon::Octagon()
    : a_(0, 0),
     b_(0, 0),
     c_(0, 0),
     d_(0, 0),
     e_(0, 0),
     f_(0, 0),
     g_(0, 0),
     h_(0, 0) {
}

Octagon::Octagon(Point a, Point b, Point c, Point d, Point e, Point f, Point g,
Point h)
    : a_(a),
     b_(b),
     c_(c),
     d_(d),
     e_(e),
     f_(f),
     g_(g),
     h_(h) {
}

std::istream& operator>>(std::istream& is, Octagon& obj) {
    is >> obj.a_;
       is >> obj.b_;
       is >> obj.c_;
       is >> obj.d_;
       is >> obj.e_;
       is >> obj.f_;
       is >> obj.g_;
       is >> obj.h_;
       return is;
}

size_t Octagon::VertexesNumber() {
    return 8;
}

std::ostream& operator<<(std::ostream& os, const Octagon& obj) {
    os << "Octagon: ";
    os <<  obj.a_ << " ";
    os <<  obj.b_ << " ";
    os <<  obj.c_ << " ";
    os <<  obj.d_ << " ";
```

```
    os <<  obj.e_ << " ";
    os <<  obj.f_ << " ";
    os <<  obj.g_ << " ";
    os <<  obj.h_ << " ";
    return os;
}

double Octagon::Area() const{
      return   0.5   *   abs(   a_.getX()*b_.getY()   +   b_.getX()*c_.getY()   +
c_.getX()*d_.getY()    +    d_.getX()*e_.getY()    +    e_.getX()*f_.getY()    +
f_.getX()*g_.getY() + g_.getX()*h_.getY() + h_.getX()*a_.getY()
      -   a_.getY()*b_.getX()   -   b_.getY()*c_.getX()   -   c_.getY()*d_.getX()   -
d_.getY()*e_.getX()    -    e_.getY()*f_.getX()    -    f_.getY()*g_.getX()    -
g_.getY()*h_.getX() - h_.getY()*a_.getX());
}

const Octagon& Octagon::operator=(const Octagon& other) {
      if (this == &other)
      return *this;

      a_ = other.a_;
      b_ = other.b_;
      c_ = other.c_;
      d_ = other.d_;
      e_ = other.e_;
      f_ = other.f_;
      g_ = other.g_;
      h_ = other.h_;

      return *this;
}

bool Octagon::operator==(const Octagon& other) const{
    if (a_ != other.a_) return false;
    if (b_ != other.b_) return false;
    if (c_ != other.c_) return false;
    if (d_ != other.d_) return false;
    if (e_ != other.e_) return false;
    if (f_ != other.f_) return false;
    if (g_ != other.g_) return false;
    if (h_ != other.h_) return false;

    return true;
}
bool Octagon::operator!=(const Octagon& other) const{
    if (a_ != other.a_) return true;
    if (b_ != other.b_) return true;
    if (c_ != other.c_) return true;
    if (d_ != other.d_) return true;
    if (e_ != other.e_) return true;
    if (f_ != other.f_) return true;
    if (g_ != other.g_) return true;
    if (h_ != other.h_) return true;

    return false;
}

Octagon::~Octagon() {
}
```

TVector.hpp

```
#ifndef TVECTOR_H
#define TVECTOR_H

#include <iostream>
#include "iterator.hpp"
#include <memory>
#define SPTR(T) std::shared_ptr<T>

template <class Polygon>
class TVector
{
public:
    // Конструктор по умолчанию
    TVector();
    // изменение размера массива
    void Resize(size_t nsize);
    // Конструктор копирования
    TVector(const TVector& other);
    // Метод, добавляющий фигуру в конец массива
    void InsertLast(const Polygon& polygon);
    // Метод, удаляющий последнюю фигуру массива
    void RemoveLast();
    // Метод, возвращающий последнюю фигуру массива
    const Polygon& Last();
    // Перегруженный оператор обращения к массиву по индексу
    const SPTR(Polygon) operator[] (const size_t idx);
    // Метод, проверяющий пустоту
    bool Empty();
    // Метод, возвращающий длину массива
    size_t Length();
    // Оператор вывода для массива в формате:
    // "[S1 S2 ... Sn]", где Si - площадь фигуры
    template <class T>
    friend std::ostream& operator<<(std::ostream& os, const TVector<T>& arr);
    // Метод, удаляющий все элементы контейнера,
    // но позволяющий пользоваться им.
    void Clear();
    // Итератор начала
    Iterator<Polygon> begin(){
        return Iterator<Polygon>(data);
    }
    // Итератор конца
    Iterator<Polygon> end(){
        return Iterator<Polygon>(data + size);
    }
    // Деструктор
    virtual ~TVector();
private:
    int size;
    SPTR(Polygon)* data;
};

#endif


template <class Polygon>
```

```
TVector<Polygon>::TVector(){
    size = 1;
    data = new SPTR(Polygon)[size];
}

template <class Polygon>
void TVector<Polygon>::Resize(size_t nsize){
       if(nsize == size)
         return;
    else{
        SPTR(Polygon)* ndata = new SPTR(Polygon)[nsize];
        for (int i = 0; i < (size < nsize ? size : nsize); i++)
            ndata[i] = data[i];
        delete[] data;
        data = ndata;
        size = nsize;
    }
}

template <class Polygon>
TVector<Polygon>::TVector(const TVector& other){
       size = other.size;
    data = new SPTR(Polygon)[other.size];
    for (int i = 0; i < size; i++)
        data[i] = other.data[i];
}

template <class Polygon>
void TVector<Polygon>::InsertLast(const Polygon& polygon){
       if (data[size - 1] != nullptr)
         Resize(size+1);
    data[size - 1] = std::make_shared<Polygon>(polygon);
}

template <class Polygon>
void TVector<Polygon>::RemoveLast(){
       data[size-1]=nullptr;
}

template <class Polygon>
const Polygon& TVector<Polygon>::Last(){
       return *(data[size - 1]);
}

template <class Polygon>
const SPTR(Polygon) TVector<Polygon>::operator[] (const size_t idx){
    if (idx >= 0 && idx < size)
        return data[idx];
    exit(1);
}

template <class Polygon>
bool TVector<Polygon>::Empty(){
       return size == 0;
}

template <class Polygon>
size_t TVector<Polygon>::Length(){
       return size;
```

```
}

template <class Polygon>
std::ostream& operator<<(std::ostream& os, const TVector<Polygon>& arr){
    os << '[';
       for (size_t i = 0; i < arr.size; i++)
        os << (arr.data[i])->Area() << ((i != arr.size-1) ? ' ' : '\0');
    os << ']';
    return os;
}

template <class Polygon>
void TVector<Polygon>::Clear(){
      delete[] data;
    size = 1;
    data = new SPTR(Polygon)[size];
}

template <class Polygon>
TVector<Polygon>::~TVector(){
    delete[] data;
}
```

Iterator.h

```
#ifndef ITERATOR_H
#define ITERATOR_H

#include <iostream>
#include <memory>

template <class Poligon>
class Iterator {
public:
    Iterator(std::shared_ptr<Poligon>* n){
        iter = n;
    }
    Poligon operator*(){
        return *(*iter);
    }
    Poligon operator->(){
        return *(*iter);
    }
    void operator++(){
        iter += 1;
    }
    Iterator operator++(int){
        Iterator iter(*this);
        ++(*this);
        return iter;
    }
    bool operator==(Iterator const& i) const{
        return iter == i.iter;
    }
    bool operator!=(Iterator const& i) const{
        return iter != i.iter;
    }

private:
    std::shared_ptr<Poligon>* iter;
```

```
};

#endif
```

main.cpp

```
#include <iostream>
#include "octagon.hpp"
#include "tvector.hpp"

int main()
{
    Octagon octi[8];
    TVector<Octagon> vec;
    for (int i = 0; i < 8; i++)
    {
        std::cin>>octi[i];
        vec.InsertLast(octi[i]);
    }
    for (auto i : vec) {
        std::cout << i << std::endl;
    }

    return 0;
}
```