МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ  
ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №7 по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Клитная Анастасия Викторовна, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

### 

### Цель работы

Целью лабораторной работы является:

Закрепление навыков работы с шаблонами классов;

Построение итераторов для динамических структур данных.

### Задание

Используя структуру данных, разработанную для лабораторной работы №4, спроектировать и разработать **итератор** для динамической структуры данных.

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:

Нельзя использовать:

Стандартные контейнеры std.

Программа должна позволять:

Вводить произвольное количество фигур и добавлять их в контейнер;

Распечатывать содержимое контейнера;

Удалять фигуры из контейнера.

**Дневник отладки**

Во время выполнения лабораторной работы возникли некоторые проблемы с реализацией списка, но все они были исправлены.

**Недочёты**  
Недочётов не было обнаружено.

**Выводы**

Лабораторная работа №7 позволила мне реализовать свой класс Iterator на языке С++, были освоены базовые навыки работы с итераторами, создаваемыми вручную, и итерирование по созданному контейнеру.

**Исходный код**

figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include "point.h"

class Figure {

public:

virtual double Area() = 0;

virtual void Print(std::ostream &os) = 0;

virtual size\_t VertexesNumber() = 0;

virtual ~Figure() {};

};

#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) {}

Point::Point(std::istream &is) {

is >> x >> y;

}

double Point::X() {

return x;

};

double Point::Y() {

return y;

};

std::istream& operator>>(std::istream& is, Point& p) {

is >> p.x >> p.y;

return is;

}

std::ostream& operator<<(std::ostream& os, Point& p) {

os << "(" << p.x << ", " << p.y << ")";

return os;

}

bool operator == (Point &p1, Point& p2) {

return (p1.x == p2.x && p1.y == p2.y);

}

Point.h

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

public:

Point();

Point(std::istream &is);

Point(double x, double y);

friend bool operator == (Point& p1, Point& p2);

friend class Pentagon;

double X();

double Y();

friend std::istream& operator>>(std::istream& is, Point& p);

friend std::ostream& operator<<(std::ostream& os, Point& p);

private:

double x;

double y;

};

#endif

## ****hlist\_item.h****

#ifndef HLISTITEM\_H

#define HLISTITEM\_H

#include <iostream>

#include "octagon.h"

#include <memory>

template <class T> class HListItem {

public:

HListItem(const std::shared\_ptr<Octagon> &octagon);

template <class A> friend std::ostream& operator<<(std::ostream& os, HListItem<A> &obj);

~HListItem();

std::shared\_ptr<T> octagon;

std::shared\_ptr<HListItem<T>> next;

std::shared\_ptr<HListItem<T>> SetNext(std::shared\_ptr<HListItem<T>> &next\_);

std::shared\_ptr<HListItem<T>> GetNext();

std::shared\_ptr<T>& GetValue();

};

#include "hlist\_item.inl"

#endif //HLISTITEM\_H

## ****hlist\_item.inl****

#include <iostream>

#include "hlist\_item.h"

template <class T> HListItem<T>::HListItem(const std::shared\_ptr<Octagon> &octagon) {

this->octagon = octagon;

this->next = nullptr;

}

template <class T> std::shared\_ptr<HListItem<T>> HListItem<T>::SetNext(std::shared\_ptr<HListItem<T>> &next\_) {

std::shared\_ptr<HListItem<T>> prev = this->next;

this->next = next\_;

return prev;

}

template <class T> std::shared\_ptr<T>& HListItem<T>::GetValue() {

return this->octagon;

}

template <class T> std::shared\_ptr<HListItem<T>> HListItem<T>::GetNext() {

return this->next;

}

template <class A> std::ostream& operator<<(std::ostream& os,HListItem<A> &obj) {

os << "[" << obj.octagon << "]" << std::endl;

return os;

}

template <class T> HListItem<T>::~HListItem() {

}

## ****main.cpp****

#include <iostream>

#include "tlinkedlist.h"

int main() {

TLinkedList<Octagon> tlinkedlist;

std::cout << tlinkedlist.Empty() << std::endl;

tlinkedlist.InsertLast(std::shared\_ptr<Octagon>(new Octagon(Point(1,2),Point(2,3),Point(3,4),Point(5,6),Point(7,8),Point(9,10), Point(11,12), Point(12,13))));

tlinkedlist.InsertLast(std::shared\_ptr<Octagon>(new Octagon(Point(13,14),Point(14,15),Point(15,16),Point(16,17),Point(17,18),Point(18,19),Point(19,20),Point(20,21))));

tlinkedlist.InsertLast(std::shared\_ptr<Octagon>(new Octagon(Point(17,18),Point(18,19),Point(19,20),Point(20,21),Point(21,22),Point(23,24), Point(25,26),Point(27,28))));

tlinkedlist.InsertLast(std::shared\_ptr<Octagon>(new Octagon(Point(17,18),Point(18,19),Point(19,20),Point(20,21),Point(21,22),Point(23,24), Point(25,26),Point(27,28))));

std::cout << tlinkedlist;

tlinkedlist.RemoveLast();

std::cout << tlinkedlist.Length() << std::endl;

tlinkedlist.RemoveFirst();

tlinkedlist.InsertFirst(std::shared\_ptr<Octagon>(new Octagon(Point(2,3),Point(3,4),Point(4,5),Point(5,6),Point(6,7),Point(7,8), Point(8,9),Point(9,10))));

tlinkedlist.Insert(std::shared\_ptr<Octagon>(new Octagon(Point(1,1),Point(2,3),Point(3,4),Point(5,6),Point(7,8),Point(9,10), Point(11,12),Point(13,18))),2);

std::cout << tlinkedlist.Empty() << std::endl;

std::cout << tlinkedlist.First() << std::endl;

std::cout << tlinkedlist.Last() << std::endl;

std::cout << tlinkedlist.GetItem(2) << std::endl;

tlinkedlist.Remove(2);

std::cout << tlinkedlist;

tlinkedlist.Clear();

return 0;

}

## ****octagon.cpp****

#include "octagon.h"

#include <cmath>

Octagon::Octagon(): point\_a(0,0), point\_b(0,0), point\_c(0,0), point\_d(0,0), point\_e(0,0), point\_f(0,0), point\_g(0,0), point\_h(0,0){

}

Octagon::Octagon(std::istream& is) {

std::cout << "Enter the octagon's vertexes:" << std::endl;

is >> point\_a;

is >> point\_b;

is >> point\_c;

is >> point\_d;

is >> point\_e;

is >> point\_f;

is >> point\_g;

is >> point\_h;

// std::cout << "The octagon is created" << std::endl;

}

Octagon::Octagon(Point point\_a1, Point point\_b1,Point point\_c1, Point point\_d1, Point point\_e1, Point point\_f1, Point point\_g1, Point point\_h1 ): point\_a(point\_a1),point\_b(point\_b1),point\_c(point\_c1),point\_d(point\_d1),point\_e(point\_e1),point\_f(point\_f1), point\_g(point\_g1),point\_h(point\_h1) {

}

/\*void Octagon::Print(std::ostream& os) {

std::cout << "Octagon: ";

std::cout << point\_a << ", ";

std::cout << point\_b << ", ";

std::cout << point\_c << ", ";

std::cout << point\_d << ", ";

std::cout << point\_e << ", ";

std::cout << point\_f << ", ";

std::cout << point\_g << ", ";

std::cout << point\_h << std::endl;

}

\*/

size\_t Octagon::VertexesNumber() {

size\_t number = 8;

return number;

}

Octagon& Octagon::operator = (const Octagon& other) {

if (this == &other) return \*this;

point\_a = other.point\_a;

point\_b = other.point\_b;

point\_c = other.point\_c;

point\_d = other.point\_d;

point\_e = other.point\_e;

point\_f = other.point\_f;

point\_g = other.point\_g;

point\_h = other.point\_h;

return \*this;

}

Octagon& Octagon::operator == (const Octagon& other) {

if (this == &other){

std::cout << "Octagons are equal" << std::endl;

} else {

std::cout << "Octagons are not equal" << std::endl;

}

}

double Octagon::Area() {

double q = abs(point\_a.X() \* point\_b.Y() + point\_b.X() \* point\_c.Y() + poiny\_c.X() \* point\_d.Y() + point\_d.X() \* point\_e.Y() + point\_e.X() \* point\_f.Y() + point\_f.X() \* point\_g.Y() + point\_g.X() \* point\_h.Y() + point\_h.X() \* point\_a.Y() - point\_b.X() \* point\_a.Y() - point\_c.X() \* point\_b.Y() - point\_d.X() \* point\_c.Y() - point\_e.X() \* point\_d.Y() - point\_f.X() \* point\_e.Y() - point\_g.X() \* point\_f.Y() - point\_h.X() \* point\_g.Y() - point\_a.X() \* point\_h.Y());

double s = q / 2;

return s;

}

Octagon::~Octagon() {

}

std::ostream& operator<<(std::ostream& os, Octagon& p) {

os << p.point\_a << p.point\_b << p.point\_c << p.point\_d << p.point\_e << p.point\_f<<p.point\_g<<p.point\_h;

return os;

}

## ****octagon.h****

#ifndef OCTAGON\_H

#define OCTAGON\_H

#include "figure.h"

class Octagon : public Figure{

public:

Octagon();

Octagon(std::istream& is);

Octagon(Point point\_a, Point point\_b, Point point\_c, Point point\_d, Point point\_e, Point point\_f, Point point\_g, Point point\_h );

size\_t VertexesNumber();

Octagon(Octagon &other);

double Area();

//void Print(std::ostream& os);

virtual ~Octagon();

Octagon& operator=(const Octagon& other);

Octagon& operator==(const Octagon& other);

friend std::ostream& operator<<(std::ostream& os, Octagon& p);

private:

Point point\_a, point\_b, point\_c, point\_d, point\_e, point\_f, point\_g, point\_h, ;

};

#endif // OCTAGON\_H

## ****titerator.h****

#include <memory>

#ifndef INC\_5\_LABA\_\_TITERATOR\_H\_

#define INC\_5\_LABA\_\_TITERATOR\_H\_

template <class node, class T> class Titerator {

public:

Titerator(std::shared\_ptr<node> n) { node\_ptr = n; }

std::shared\_ptr<T> operator\*() { return node\_ptr->GetValue(); }

std::shared\_ptr<T> operator->() { return node\_ptr->GetValue(); }

void operator++() { node\_ptr = node\_ptr->GetNext(); }

Titerator operator++(int) {

Titerator other(\*this);

++(\*this);

return other;

}

bool operator==(Titerator const &i) { return node\_ptr == i.node\_ptr; };

bool operator!=(Titerator const &i) { return node\_ptr != i.node\_ptr; };

private:

std::shared\_ptr<node> node\_ptr;

};

#endif // INC\_5\_LABA\_\_TITERATOR\_H\_

## ****tlinkedlist.h****

#ifndef HLIST\_H

#define HLIST\_H

#include <iostream>

#include "hlist\_item.h"

#include "octagon.h"

#include <memory>

#include "titerator.h"

template <class T> class TLinkedList {

public:

TLinkedList();

int size\_of\_list;

size\_t Length();

std::shared\_ptr<T>& First();

std::shared\_ptr<Octagon>& Last();

std::shared\_ptr<Octagon>& GetItem(size\_t idx);

bool Empty();

TLinkedList(const std::shared\_ptr<TLinkedList> &other);

void InsertFirst(const std::shared\_ptr<Octagon> &&octagon);

void InsertLast(const std::shared\_ptr<Octagon> &&octagon);

void RemoveLast();

void RemoveFirst();

void Insert(const std::shared\_ptr<Octagon> &&octagon, size\_t position);

void Remove(size\_t position);

void Clear();

template <class A> friend std::ostream& operator<<(std::ostream& os, TLinkedList<A>& list);

~TLinkedList();

Titerator<HListItem<T>, T> begin();

Titerator<HListItem<T>, T> end();

private:

std::shared\_ptr<HListItem<T>> front;

std::shared\_ptr<HListItem<T>> back;

};

#include "tlinkedlist.inl"

#endif //HList\_H

## ****tlinkedlist.inl****

#include <iostream>

#include "tlinkedlist.h"

template <class T>

Titerator<HListItem<T>, T> TLinkedList<T>::begin() {

return Titerator<HListItem<T>, T> (front);

}

template <class T>

Titerator<HListItem<T>, T> TLinkedList<T>::end() {

return Titerator<HListItem<T>, T>(back);

}

template <class T> TLinkedList<T>::TLinkedList() {

size\_of\_list = 0;

std::shared\_ptr<HListItem<T>> front = nullptr;

std::shared\_ptr<HListItem<T>> back = nullptr;

std::cout << "Octagon List created" << std::endl;

}

template <class T> TLinkedList<T>::TLinkedList(const std::shared\_ptr<TLinkedList> &other){

front = other->front;

back = other->back;

}

template <class T> size\_t TLinkedList<T>::Length() {

return size\_of\_list;

}

template <class T> bool TLinkedList<T>::Empty() {

return size\_of\_list;

}

template <class T> std::shared\_ptr<Hexagon>& TLinkedList<T>::GetItem(size\_t idx){

int k = 0;

std::shared\_ptr<HListItem<T>> obj = front;

while (k != idx){

k++;

obj = obj->GetNext();

}

return obj->GetValue();

}

template <class T> std::shared\_ptr<T>& TLinkedList<T>::First() {

return front->GetValue();

}

template <class T> std::shared\_ptr<Hexagon>& TLinkedList<T>::Last() {

return back->GetValue();

}

template <class T> void TLinkedList<T>::InsertLast(const std::shared\_ptr<Octagon> &&octagon) {

std::shared\_ptr<HListItem<T>> obj (new HListItem<T>(octagon));

// std::shared\_ptr<HListItem<T>> obj = std::make\_shared<HListItem<T>>(HListItem<T>(octagon));

if(size\_of\_list == 0) {

front = obj;

back = obj;

size\_of\_list++;

return;

}

back->SetNext(obj); // = obj;

back = obj;

obj->next = nullptr; // = nullptr;

size\_of\_list++;

}

template <class T> void TLinkedList<T>::RemoveLast() {

if (size\_of\_list == 0) {

std::cout << "Octagon does not pop\_back, because the Octagon List is empty" << std:: endl;

} else {

if (front == back) {

RemoveFirst();

size\_of\_list--;

return;

}

std::shared\_ptr<HListItem<T>> prev\_del = front;

while (prev\_del->GetNext() != back) {

prev\_del = prev\_del->GetNext();

}

prev\_del->next = nullptr;

back = prev\_del;

size\_of\_list--;

}

}

template <class T> void TLinkedList<T>::InsertFirst(const std::shared\_ptr<Octagon> &&octagon) {

std::shared\_ptr<HListItem<T>> obj (new HListItem<T>(octagon));

if(size\_of\_list == 0) {

front = obj;

back = obj;

} else {

obj->SetNext(front); // = front;

front = obj;

}

size\_of\_list++;

}

template <class T> void TLinkedList<T>::RemoveFirst() {

if (size\_of\_list == 0) {

std::cout << "Octagon does not pop\_front, because the Octagon List is empty" << std:: endl;

} else {

std::shared\_ptr<HListItem<T>> del = front;

front = del->GetNext();

size\_of\_list--;

}

}

template <class T> void TLinkedList<T>::Insert(const std::shared\_ptr<Octagon> &&octagon,size\_t position) {

if (position <0) {

std::cout << "Position < zero" << std::endl;

} else if (position > size\_of\_list) {

std::cout << " Position > size\_of\_list" << std::endl;

} else {

std::shared\_ptr<HListItem<T>> obj (new HListItem<T>(octagon));

if (position == 0) {

front = obj;

back = obj;

} else {

int k = 0;

std::shared\_ptr<HListItem<T>> prev\_insert = front;

std::shared\_ptr<HListItem<T>> next\_insert;

while(k+1 != position) {

k++;

prev\_insert = prev\_insert->GetNext();

}

next\_insert = prev\_insert->GetNext();

prev\_insert->SetNext(obj); // = obj;

obj->SetNext(next\_insert); // = next\_insert;

}

size\_of\_list++;

}

}

template <class T> void TLinkedList<T>::Remove(size\_t position) {

if (position > size\_of\_list ) {

std:: cout << "Position " << position << " > " << "size " << size\_of\_list << " Not correct erase" << std::endl;

} else if (position < 0) {

std::cout << "Position < 0" << std::endl;

} else {

if (position == 0) {

RemoveFirst();

} else {

int k = 0;

std::shared\_ptr<HListItem<T>> prev\_erase = front;

std::shared\_ptr<HListItem<T>> next\_erase;

std::shared\_ptr<HListItem<T>> del;

while( k+1 != position) {

k++;

prev\_erase = prev\_erase->GetNext();

}

next\_erase = prev\_erase->GetNext();

del = prev\_erase->GetNext();

next\_erase = del->GetNext();

prev\_erase->SetNext(next\_erase); // = next\_erase;

}

size\_of\_list--;

}

}

template <class T> void TLinkedList<T>::Clear() {

std::shared\_ptr<HListItem<T>> del = front;

std::shared\_ptr<HListItem<T>> prev\_del;

if(size\_of\_list !=0 ) {

while(del->GetNext() != nullptr) {

prev\_del = del;

del = del->GetNext();

}

size\_of\_list = 0;

// std::cout << "HListItem deleted" << std::endl;

}

size\_of\_list = 0;

std::shared\_ptr<HListItem<T>> front;

std::shared\_ptr<HListItem<T>> back;

}

template <class T> std::ostream& operator<<(std::ostream& os, TLinkedList<T>& ol) {

if (ol.size\_of\_list == 0) {

os << "The octagon list is empty, so there is nothing to output" << std::endl;

} else {

os << "Print Octagon List" << std::endl;

std::shared\_ptr<HListItem<T>> obj = ol.front;

while(obj != nullptr) {

if (obj->GetNext() != nullptr) {

os << obj->GetValue() << " " << "," << " ";

obj = obj->GetNext();

} else {

os << obj->GetValue();

obj = obj->GetNext();

}

}

os << std::endl;

}

return os;

}

template <class T> TLinkedList<T>::~TLinkedList() {

std::shared\_ptr<HListItem<T>> del = front;

std::shared\_ptr<HListItem<T>> prev\_del;

if(size\_of\_list !=0 ) {

while(del->GetNext() != nullptr) {

prev\_del = del;

del = del->GetNext();

}

size\_of\_list = 0;

std::cout << "Octagon List deleted" << std::endl;

}

}