МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №7**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студентка: *Варламова Анна Борисовна*

Группа:  *М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

Москва, 2021

**Задание:**

Используя структуры данных, разработанные для лабораторной работы №6, спроектировать и разработать итератор для динамической структуры данных.

Итератор должен быть разработан в виде шаблона, должен работать со всеми типами фигур, согласно варианту задания.

Итератор должен позволять использовать структуру данных в операторах типа for, например:

for (auto i : list) {

std::cout << \*i << std::endl;

}

**Вариант №8:**

* + Фигура: Восьмиугольник (Octagon)
  + Контейнер: Список (TLinkedList)

**Описание программы:**

Исходный код разделён на 11 файлов:

* figure.h – описание класса фигуры
* point.h – описание класса точки
* point.cpp – реализация класса точки
* octagon.h – описание класса восьмиугольника
* octagon.cpp – реализация класса восьмиугольника
* item.h – описание элемента списка
* item.cpp – реализация элемента списка
* tlinkedlist.h – описание списка
* tlinkedlist.cpp – реализация списка
* main.cpp – основная программа
* titerator.h – описание и реализация итератора

**Дневник отладки:**

Я забыла сделать перегрузку оператора ++ типа void, из-за чего не работал цикл с auto. Проблема была решена.

**Вывод:**  
 При выполнении задания я на практике освоила основы реализации и работы с итераторами. Они позволяют удобно обходить контейнер через циклы, перебирая элементы. Навык работы с итераторами крайне полезен, он позволяет поднять уровень абстракции, при использовании контейнера необязательно разбираться, как он устроен.

**Исходный код:**

**figure.h**

#ifndef FIGURE\_H

#define FIGURE\_H

#include "point.h"

class Figure

{

public:

virtual size\_t VertexesNumber() = 0;

virtual double Area() = 0;

virtual ~Figure(){};

};

#endif // FIGURE\_H

**point.h:**

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

public:

Point();

Point(std::istream &is);

Point(double x, double y);

double dist(Point& other);

bool operator==(const Point &other);

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

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

private:

double x\_;

double y\_;

};

#endif // POINT\_H

**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::dist(Point& other)

{

double dx = (other.x\_ - x\_);

double dy = (other.y\_ - y\_);

return std::sqrt(dx\*dx + dy\*dy);

}

bool Point::operator==(const Point &other)

{

return ((x\_ == other.x\_) && (y\_ == other.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;

}

**octagon.h:**

#ifndef OCTAGON\_H

#define OCTAGON\_H

#include "figure.h"

class Octagon : public Figure

{

public:

Octagon();

Octagon(Point t\_1, Point t\_2, Point t\_3, Point t\_4,

Point t\_5, Point t\_6, Point t\_7, Point t\_8);

Octagon(const Octagon& other);

Octagon(std::istream &is);

Octagon &operator=(const Octagon &other);

bool operator==(const Octagon &other);

friend std::istream& operator>>(std::istream& is, Octagon& o);

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

double Area();

size\_t VertexesNumber();

virtual ~Octagon();

private:

Point t1;

Point t2;

Point t3;

Point t4;

Point t5;

Point t6;

Point t7;

Point t8;

};

#endif // OCTAGON\_H

**octagon.cpp:**

#include "octagon.h"

#include <iostream>

#include <cmath>

Octagon::Octagon()

: t1(0.0, 0.0), t2(0.0, 0.0), t3(0.0, 0.0), t4(0.0, 0.0),

t5(0.0, 0.0), t6(0.0, 0.0), t7(0.0, 0.0), t8(0.0, 0.0) {}

Octagon::Octagon(Point t\_1, Point t\_2, Point t\_3, Point t\_4,

Point t\_5, Point t\_6, Point t\_7, Point t\_8)

: t1(t\_1), t2(t\_2), t3(t\_3), t4(t\_4),

t5(t\_5), t6(t\_6), t7(t\_7), t8(t\_8) {}

Octagon::Octagon(const Octagon& other)

: Octagon(other.t1, other.t2, other.t3, other.t4,

other.t5, other.t6, other.t7, other.t8) {}

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

{

is >> t1 >> t2 >> t3 >> t4 >> t5 >> t6 >> t7 >> t8;

}

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

{

if (this == &other) {

return \*this;

}

t1 = other.t1;

t2 = other.t2;

t3 = other.t3;

t4 = other.t4;

t5 = other.t5;

t6 = other.t6;

t7 = other.t7;

t8 = other.t8;

return \*this;

}

bool Octagon::operator==(const Octagon &o)

{

if ((t1 == o.t1) && (t2 == o.t2) && (t3 == o.t3) && (t4 == o.t4) &&

(t5 == o.t5) && (t6 == o.t6) && (t7 == o.t7) && (t8 == o.t8))

return true;

else

return false;

}

std::istream& operator>>(std::istream& is, Octagon& o)

{

is >> o.t1 >> o.t2 >> o.t3 >> o.t4 >> o.t5 >> o.t6 >> o.t7 >> o.t8;

return is;

}

std::ostream& operator<<(std::ostream& os, const Octagon& o)

{

os << "Octagon: " << o.t1 << " " << o.t2 << " " << o.t3 << " " << o.t4

<< " " << o.t5 << " " << o.t6 << " " << o.t7 << " " << o.t8;

return os;

}

size\_t Octagon::VertexesNumber()

{

return (size\_t)8;

}

double Heron(Point A, Point B, Point C)

{

double AB = A.dist(B);

double BC = B.dist(C);

double AC = A.dist(C);

double p = (AB + BC + AC) / 2;

return sqrt(p \* (p - AB) \* (p - BC) \* (p - AC));

}

double Octagon::Area()

{

double area1 = Heron(t1, t2, t3);

double area2 = Heron(t1, t4, t3);

double area3 = Heron(t1, t4, t5);

double area4 = Heron(t1, t5, t6);

double area5 = Heron(t1, t6, t7);

double area6 = Heron(t1, t7, t8);

return area1 + area2 + area3 + area4 + area5 + area6;

}

Octagon::~Octagon() {}

**item.h:**

#ifndef ITEM\_H

#define ITEM\_H

#define tT template <typename T>

#define sIT std::shared\_ptr<Item<T>>

#define sT std::shared\_ptr<T>

#define IT Item<T>

#define sI std::shared\_ptr<Item>

#include "octagon.h"

#include <memory>

#include <iostream>

tT

class Item

{

public:

Item(const sT &s);

Item(const Item &other);

sI Left();

sI Right();

void ToLeft(sI node);

void ToRight(sI node);

sT GetOctagon() const;

template <class O>

friend std::ostream &operator<<(std::ostream &os, const Item<O> &node);

virtual ~Item();

private:

sT octagon;

sIT prev;

sIT next;

};

#endif // ITEM\_H

**item.cpp:**

#include "item.h"

tT

IT::Item(const sT &o)

{

this->octagon = o;

this->next = nullptr;

this->prev = nullptr;

}

tT

IT::Item(const IT &other)

{

this->octagon = other.octagon;

this->next = other.next;

this->prev = other.prev;

}

tT

sIT IT::Left()

{

return this->prev;

}

tT

sIT IT::Right()

{

return this->next;

}

tT

void IT::ToLeft(sIT node)

{

this->prev = node;

}

tT

void IT::ToRight(sIT node)

{

this->next = node;

}

tT

sT IT::GetOctagon() const

{

return this->octagon;

}

tT

std::ostream &operator<<(std::ostream &os, const IT& node)

{

os << node.octagon << std::endl;

return os;

}

tT

IT::~Item() {}

template class Item<Octagon>;

template std::ostream& operator<<(std::ostream& os, const Item<Octagon>& item);

**tlinkedlist.h:**

#ifndef TLINKEDLIST\_H

#define TLINKEDLIST\_H

#define LT TLinkedList<T>

#include "item.h"

tT

class TLinkedList

{

public:

TLinkedList();

TLinkedList(const LT &other);

sT First();

sT Last();

sT GetItem(size\_t idx);

size\_t Length();

bool Empty();

void InsertFirst(sT octagon);

void InsertLast(sT octagon);

void Insert(sT octagon, size\_t position);

void RemoveFirst();

void RemoveLast();

void Remove(size\_t position);

template <class I>

friend std::ostream &operator<<(std::ostream &os, const TLinkedList<I> &list);

void Clear();

virtual ~TLinkedList();

private:

sIT beginning;

sIT end;

};

#endif // TLINKEDLIST\_H

**tlinkedlist.cpp:**

#include "tlinkedlist.h"

tT

LT::TLinkedList() : beginning(nullptr), end(nullptr) {}

tT

LT::TLinkedList(const TLinkedList &other)

{

beginning = other.beginning;

end = other.end;

}

tT

sT LT::First()

{

if (beginning == nullptr) {

std::cout << "The list is empty" << std::endl;

exit(1);

}

return beginning->GetOctagon();

}

tT

sT LT::Last()

{

if (end == nullptr) {

std::cout << "The list is empty" << std::endl;

exit(1);

}

return end->GetOctagon();

}

tT

sT LT::GetItem(size\_t position)

{

size\_t n = this->Length();

if (beginning == nullptr) {

std::cout << "The list is empty" << std::endl;

exit(1);

}

if (position > n) {

std::cout << "The is no such position" << std::endl;

exit(1);

}

if (position == 1) {

return beginning->GetOctagon();

}

if (position == n) {

return end->GetOctagon();

}

sIT node = beginning;

for (size\_t i = 1; i < position; ++i) {

node = node->Right();

}

return node->GetOctagon();

}

tT

bool LT::Empty()

{

return (beginning == nullptr);

}

tT

size\_t LT::Length()

{

size\_t size = 0;

for (sIT i = beginning; i != nullptr; i = i->Right()) {

++size;

}

return size;

}

tT

void LT::InsertFirst(sT octagon)

{

sIT node(new IT(octagon));

if (beginning == nullptr) {

beginning = (end = node);

return;

}

node->ToLeft(nullptr);

node->ToRight(beginning);

beginning->ToLeft(node);

beginning = node;

}

tT

void LT::InsertLast(sT octagon)

{

sIT node(new IT(octagon));

if (beginning == nullptr) {

beginning = (end = node);

return;

}

node->ToLeft(end);

node->ToRight(nullptr);

end->ToRight(node);

end = node;

}

tT

void LT::Insert(sT octagon, size\_t position)

{

size\_t n = this->Length();

if (position > n + 1) {

std::cout << "The is no such position" << std::endl;

return;

}

if (position == 1) {

InsertFirst(octagon);

return;

}

if (position == n + 1) {

InsertLast(octagon);

return;

}

sIT node(new IT(octagon));

sIT now = beginning;

for (size\_t i = 1; i < position; ++i) {

now = now->Right();

}

sIT before = now->Left();

before->ToRight(node);

now->ToLeft(node);

node->ToLeft(before);

node->ToRight(now);

}

tT

void LT::RemoveFirst()

{

if (beginning == nullptr) {

std::cout << "The list is empty" << std::endl;

return;

}

if (end == beginning) {

beginning = (end = nullptr);

return;

}

sIT node = beginning;

beginning = beginning->Right();

beginning->ToLeft(nullptr);

}

tT

void LT::RemoveLast()

{

if (end == nullptr) {

std::cout << "The list is empty" << std::endl;

return;

}

if (end == beginning) {

beginning = (end = nullptr);

return;

}

sIT node = end;

end = end->Left();

end->ToRight(nullptr);

}

tT

void LT::Remove(size\_t position)

{

size\_t n = this->Length();

if (beginning == nullptr) {

std::cout << "The list is empty" << std::endl;

return;

}

if (position > n) {

std::cout << "The is no such position" << std::endl;

return;

}

if (position == 1) {

RemoveFirst();

return;

}

if (position == n) {

RemoveLast();

return;

}

sIT node = beginning;

for (size\_t i = 1; i < position; ++i) {

node = node->Right();

}

sIT node\_left = node->Left();

sIT node\_right = node->Right();

node\_left->ToRight(node\_right);

node\_right->ToLeft(node\_left);

}

tT

std::ostream &operator<<(std::ostream &os, const LT &list)

{

if (list.beginning == nullptr) {

os << "List is empty" << std::endl;

return os;

}

for (sIT i = list.beginning; i != nullptr; i = i->Right()) {

if (i->Right() != nullptr)

os << i->GetOctagon()->Area() << " -> ";

else

os << i->GetOctagon()->Area();

}

return os;

}

tT

void LT::Clear()

{

while (beginning != nullptr) {

RemoveFirst();

}

}

tT

LT::~TLinkedList()

{

while (beginning != nullptr) {

RemoveFirst();

}

}

template class TLinkedList<Octagon>;

template std::ostream& operator<<(std::ostream& os, const TLinkedList<Octagon>& list);

**titerator.h:**

#ifndef TITERATOR\_H

#define TITERATOR\_H

#include <iostream>

#include <memory>

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->GetOctagon();

}

std::shared\_ptr<T> operator->()

{

return node\_ptr->GetOctagon();

}

// --i ++i

TIterator operator++(int)

{

TIterator iter(\*this);

++(\*this);

return iter;

}

TIterator operator--(int)

{

TIterator iter(\*this);

--(\*this);

return iter;

}

// i++ i--

void operator++()

{

node\_ptr = node\_ptr->Right();

}

void operator--()

{

node\_ptr = node\_ptr->Left();

}

bool operator==(TIterator const &i)

{

return node\_ptr == i.node\_ptr;

}

bool operator!=(TIterator const &i)

{

return !(\*this == i);

}

private:

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

};

#endif // TITERATOR\_H

**main.cpp:**

#include "tlinkedlist.h"

int main(void)

{

TLinkedList<Octagon> l;

Point x1(3, 1);

Point x2(2, 4);

Point x3(4, 8);

Point x4(7, 8);

Point x5(9, 6);

Point x6(10, 3);

Point x7(9, 1);

Point x8(6, 0);

Point y1(3, 0);

Point y2(1, 2);

Point y3(1, 4);

Point y4(3, 5);

Point y5(5, 5);

Point y6(7, 4);

Point y7(7, 2);

Point y8(5, 0);

Point z1(2, 0);

Point z2(1, 2);

Point z3(1, 5);

Point z4(5, 6);

Point z5(6, 5);

Point z6(7, 3);

Point z7(6, 1);

Point z8(4, 0);

std::shared\_ptr<Octagon> o1(new Octagon(x1, x2, x3, x4, x5, x6, x7, x8));

std::shared\_ptr<Octagon> o2(new Octagon(y1, y2, y3, y4, y5, y6, y7, y8));

std::shared\_ptr<Octagon> o3(new Octagon(z1, z2, z3, z4, z5, z6, z7, z8));

*/\*std::shared\_ptr<Octagon> o1(new Octagon);*

*std::shared\_ptr<Octagon> o2(new Octagon);*

*std::shared\_ptr<Octagon> o3(new Octagon);*

*std::cin >> \*o1 >> \*o2 >> \*o3;\*/*

l.Remove(5);

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

l.Insert(o1, 1);

std::cout << l << std::endl;

l.Insert(o1, 2);

std::cout << l << std::endl;

l.Insert(o1, 3);

std::cout << l << std::endl;

l.Insert(o3, 4);

std::cout << l << std::endl;

l.Insert(o3, 3);

std::cout << l << std::endl;

l.Insert(o2, 6);

std::cout << l << std::endl;

l.Insert(o2, 1);

std::cout << l << std::endl;

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

std::cout << std::endl;

for (auto i : l) {

std::cout << \*i << std::endl;

}

std::cout << std::endl;

l.Remove(5);

std::cout << l << std::endl;

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

l.Remove(l.Length());

std::cout << l << std::endl;

l.RemoveFirst();

std::cout << l << std::endl;

l.RemoveLast();

std::cout << l << std::endl;

l.InsertFirst(o3);

std::cout << l << std::endl;

std::cout << std::endl;

for (auto i : l) {

std::cout << \*i << std::endl;

}

std::cout << std::endl;

std::cout << \*l.GetItem(1) << std::endl;

std::cout << \*l.GetItem(2) << std::endl;

std::cout << \*l.GetItem(3) << std::endl;

std::cout << \*l.GetItem(4) << std::endl;

std::cout << \*l.Last() << std::endl;

return 0;

}