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

ЛАБОРАТОРНАЯ РАБОТА №5

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:

Дополнить класс-контейнер из лабораторной работы №4 умными указателями.

Вариант №8:

- Фигура: Восьмиугольник (Octagon)
- Контейнер: Список (TLinkedList)

Описание программы:

Исходный код разделён на 10 файлов:

- point.h – описание класса точки
- point.cpp – реализация класса точки
- figure.h – описание класса фигуры
- 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

Дневник отладки:

Было неочевидно, что в функции (методы) надо передавать константы, из-за этого возникали ошибки. Исправлено припиской const.

Вывод:

При выполнении задания я на практике освоила основы работы с умными указателями. Они позволяют избежать проблем с утечками памяти, разыменовыванием нулевого указателя (или с обращением к неинициализированной области памяти), а также с удалением уже удалённого объекта.

Исходный код:

figure.h

```
1. #ifndef FIGURE_H
2. #define FIGURE_H
3.
4. #include "point.h"
5.
```

```
class Figure {
public:
  virtual size_t VertexesNumber() = 0;
  virtual void Print(std::ostream& os) = 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);

  friend std::istream& operator>>(std::istream& is, Point& p);
  friend std::ostream& operator<<(std::ostream& os, 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);
```

```
}

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;
}
```

octagon.h

```
#ifndef OCTAGON_H
#define OCTAGON_H

#include <iostream>

#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(std::istream &is);
  Octagon(const Octagon& other);

  size_t VertexesNumber();
  double Area();
  void Print(std::ostream& os);

  virtual ~Octagon();

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

private:
  Point t1;
  Point t2;
  Point t3;
  Point t4;
  Point t5;
  Point t6;
  Point t7;
  Point t8;
};

#endif // OCTAGON_H
```

octagon.cpp

```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(std::istream &is) {
  is >> t1 >> t2 >> t3 >> t4 >> t5 >> t6 >> t7 >> t8;
}

Octagon::Octagon(const Octagon& other)
  : Octagon(other.t1, other.t2, other.t3, other.t4,
            other.t5, other.t6, other.t7, other.t8) {}

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, 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);
```

```
54.    return area1 + area2 + area3 + area4 + area5 + area6;
55.  }
56.
57.  void Octagon::Print(std::ostream& os)
58.  {
59.    std::cout << "Octagon: " << t1 << " " << t2 << " " << t3 << " " << t4
60.              << " " << t5 << " " << t6 << " " << t7 << " " << t8 << "\n";
61.  }
62.
63.  Octagon::~Octagon() {}
```

item.h:

```
#ifndef ITEM_H
#define ITEM_H

#include "octagon.h"
#include <memory>

#define ShOct std::shared_ptr<Octagon>
#define ShItem std::shared_ptr<Item>

class Item
{
public:
    Item(const ShOct &s);
    Item(const Item &other);

    ShItem Left();
    ShItem Right();

    void ToLeft(ShItem node);
    void ToRight(ShItem node);

    ShOct GetOctagon();

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

    virtual ~Item();

private:
    ShOct octagon;
    ShItem prev;
    ShItem next;
};

#endif // ITEM_H
```

item.cpp:

```
#include "item.h"

Item::Item(const ShOct &o)
{
    this->octagon = o;
    this->next = nullptr;
    this->prev = nullptr;
}
```

```
Item::Item(const Item &other)
{
   this->octagon = other.octagon;
   this->next = other.next;
   this->prev = other.prev;
}

ShItem Item::Left()
{
   return this->prev;
}

ShItem Item::Right()
{
   return this->next;
}

void Item::ToLeft(ShItem node)
{
   this->prev = node;
}

void Item::ToRight(ShItem node)
{
   this->next = node;
}

ShOct Item::GetOctagon()
{
   return this->octagon;
}

std::ostream &operator<<(std::ostream &os, const Item &node)
{
   os << node.octagon << std::endl;
   return os;
}

Item::~Item() {}
```

tlinkedlist.h:

```
#include "item.h"

Item::Item(const ShOct &o)
{
   this->octagon = o;
   this->next = nullptr;
   this->prev = nullptr;
}

Item::Item(const Item &other)
{
   this->octagon = other.octagon;
   this->next = other.next;
   this->prev = other.prev;
}

ShItem Item::Left()
```

```
{
    return this->prev;
}

ShItem Item::Right()
{
    return this->next;
}

void Item::ToLeft(ShItem node)
{
    this->prev = node;
}

void Item::ToRight(ShItem node)
{
    this->next = node;
}

ShOct Item::GetOctagon()
{
    return this->octagon;
}

std::ostream &operator<<(std::ostream &os, const Item &node)
{
    os << node.octagon << std::endl;
    return os;
}

Item::~Item() {}
```

tlinkedlist.cpp:

```
#include "tlinkedlist.h"

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

TLinkedList::TLinkedList(const TLinkedList &other)
{
    beginning = other.beginning;
    end = other.end;
}

ShOct TLinkedList::First()
{
    if (beginning == nullptr) {
        std::cout << "The list is empty" << std::endl;
        exit(1);
    }
    return beginning->GetOctagon();
}

ShOct TLinkedList::Last()
{
    if (end == nullptr) {
        std::cout << "The list is empty" << std::endl;
        exit(1);
    }
```

```
    return end->GetOctagon();
}

ShOct TLinkedList::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();
    }
    ShItem node = beginning;
    for (size_t i = 1; i < position; ++i) {
        node = node->Right();
    }
    return node->GetOctagon();
}

bool TLinkedList::Empty()
{
    return (beginning == nullptr);
}

size_t TLinkedList::Length()
{
    size_t size = 0;
    for (ShItem i = beginning; i != nullptr; i = i->Right()) {
        ++size;
    }
    return size;
}

void TLinkedList::InsertFirst(ShOct octagon)
{
    ShItem node(new Item(octagon));
    if (beginning == nullptr) {
        beginning = (end = node);
        return;
    }
    node->ToLeft(nullptr);
    node->ToRight(beginning);
    beginning->ToLeft(node);
    beginning = node;
}

void TLinkedList::InsertLast(ShOct octagon)
{
    ShItem node(new Item(octagon));
    if (beginning == nullptr) {
        beginning = (end = node);
        return;
```

```
    }
    node->ToLeft(end);
    node->ToRight(nullptr);
    end->ToRight(node);
    end = node;
}

void TLinkedList::Insert(ShOct 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;
    }
    ShItem node(new Item(octagon));
    ShItem now = beginning;
    for (size_t i = 1; i < position; ++i) {
        now = now->Right();
    }
    ShItem before = now->Left();
    before->ToRight(node);
    now->ToLeft(node);
    node->ToLeft(before);
    node->ToRight(now);
}

void TLinkedList::RemoveFirst()
{
    if (beginning == nullptr) {
        std::cout << "The list is empty" << std::endl;
        return;
    }
    if (end == beginning) {
        beginning = (end = nullptr);
        return;
    }
    ShItem node = beginning;
    beginning = beginning->Right();
    beginning->ToLeft(nullptr);
}

void TLinkedList::RemoveLast()
{
    if (end == nullptr) {
        std::cout << "The list is empty" << std::endl;
        return;
    }
    if (end == beginning) {
        beginning = (end = nullptr);
        return;
    }
    ShItem node = end;
```

```
    end = end->Left();
    end->ToRight(nullptr);
}

void TLinkedList::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;
    }
    ShItem node = beginning;
    for (size_t i = 1; i < position; ++i) {
        node = node->Right();
    }
    ShItem node_left = node->Left();
    ShItem node_right = node->Right();
    node_left->ToRight(node_right);
    node_right->ToLeft(node_left);
}

std::ostream &operator<<(std::ostream &os, const TLinkedList &list)
{
    if (list.beginning == nullptr) {
        os << "List is empty" << std::endl;
        return os;
    }
    for (ShItem i = list.beginning; i != nullptr; i = i->Right()) {
        if (i->Right() != nullptr)
            os << i->GetOctagon()->Area() << " -> ";
        else
            os << i->GetOctagon()->Area();
    }
    return os;
}

void TLinkedList::Clear()
{
    while (beginning != nullptr) {
        RemoveFirst();
    }
}

TLinkedList::~TLinkedList()
{
    while (beginning != nullptr) {
        RemoveFirst();
    }
```

```
}
```

main.cpp:

```
#include "tlinkedlist.h"

int main(void)
{
    TLinkedList 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);

    ShOct o1(new Octagon(x1, x2, x3, x4, x5, x6, x7, x8));
    ShOct o2(new Octagon(y1, y2, y3, y4, y5, y6, y7, y8));
    ShOct o3(new Octagon(z1, z2, z3, z4, z5, z6, z7, z8));

    /*ShOct o1(new Octagon);
    ShOct o2(new Octagon);
    ShOct o3(new Octagon);
    std::cin >> *o1 >> *o2 >> *o3;*/
    l.Remove(5);
    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;
```

```
    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 << *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;
    return 0;
}
```