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

# ЛАБОРАТОРНАЯ РАБОТА №4

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Белоусов Егор Владимирович, группа М8О-207Б-20*

Преподаватель *Дорохов Евгений Павлович*

**Цель работы**

Целью лабораторной работы является:

- Закрепление навыков работы с классами.
- Создание простых динамических структур данных.
- Работа с объектами, передаваемыми «по значению».

**Задание**

Вариант 7: связанный список (TLinkedList)

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий шестиугольник**,** согласно вариантам задания. Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Классы фигур должны содержать набор следующих методов:
  - Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>). Он должен заменить конструктор, принимающий координаты вершин из стандартного потока.
  - Перегруженный оператор вывода в поток std::ostream (<<), заменяющий метод Print из лабораторной работы 1.
  - Оператор копирования (=)
  - Оператор сравнения с такими же фигурами (==)
- Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
- Класс-контейнер должен содержать набор следующих методов:
  - Метод по добавлению фигуры в контейнер.
    - Связанный список: **InsertFirst, InsertLast, Insert**
  - Метод по получению фигуры из контейнера.
    - Связанный список: **First, Last, GetItem**
  - Метод по удалению фигуры из контейнера.
    - Связанный список: **RemoveFirst, RemoveLast, Remove**
  - Перегруженный оператор по выводу контейнера в поток std::ostream (<<).
  - Деструктор, удаляющий все элементы контейнера.
  - Набор специальных методов для класса-контейнера.

## Описание программы

Исходный код лежит в 10 файлах:

1. main.cpp основная программа

2. hexagon.h описание класса шестиугольник

3. hexagon.cpp реализация класса шестиугольник

4. point.h описание класса точка

5. point.cpp реализация класса точка

6. tlinkedlist_item.h описание класса-элемента списка

7. tlinkedlist_item.cpp реализация класса-элемента списка

8. tlinkedlist.h описание класса списка

9. tlinkedlist.cpp реализация класса списка

## Дневник отладки

*D:\oop\lab2\cmake-build-debug\lab2.exe*

*0 0 0 2 2 3 4 2 4 0 2 -1*

*List:*

*Hexagon: (0, 0) (0, 2) (2, 3) (4, 2) (4, 0) (2, -1)*

*1*

*1 2 3 4 5 6 7 8 9 10 11 12*

*List:*

*Hexagon: (1, 2) (3, 4) (5, 6) (7, 8) (9, 10) (11, 12)*

*Hexagon: (0, 0) (0, 2) (2, 3) (4, 2) (4, 0) (2, -1)*

*2*

*DELETED ITEM*

*List:*

*Hexagon: (1, 2) (3, 4) (5, 6) (7, 8) (9, 10) (11, 12)*

*1*

*DELETED ITEM*

*List:*

*0*

*DELETED LIST*

*Process finished with exit code 0*

## Недочёты

По моему мнению недочетов нет.

## Выводы

В данной лабораторной работе я закрепил свои знания, полученные из первой лабораторной работы, создал простейшую динамическую структуру данных: связанный список, используя непосредственно ООП. Работал с объектами, передавая их по значению.

## Исходный код

main.cpp

```
#include "tlinkedlist.h"

using std::cin;
using std::cout;

int main() {
    TLinkedList list;
    Hexagon hex;
    cin >> hex;
    list.InsertFirst(hex);
    cout << list;
    cout << list.Length() << "\n";
    cin >> hex;
    list.InsertFirst(hex);
    cout << list;
    cout << list.Length() << "\n";
    list.RemoveLast();
    cout << list;
    cout << list.Length() << "\n";
    list.RemoveFirst();
    cout << list;
    cout << list.Length() << "\n";
    return 0;
}
```

hexagon.h

```h
#ifndef LAB2_HEXAGON_H
#define LAB2_HEXAGON_H


#include "iostream"
#include "point.h"

class Hexagon {
private:
    static const size_t size = 6;
    Point P[size];
public:
    Hexagon();

    double Area();

    size_t VertexesNumber();

    Hexagon &operator=(const Hexagon &other);

    friend bool operator==(const Hexagon &a, const Hexagon &b);

    friend std::istream &operator>>(std::istream &is, Hexagon &hexagon);

    friend std::ostream &operator<<(std::ostream &os, const Hexagon &hexagon);

    ~Hexagon();
};


#endif //LAB2_HEXAGON_H
```

## hexagon.cpp

```cpp
#include "hexagon.h"


Hexagon::Hexagon() {}


double Hexagon::Area() {
    const double eps = 1e-9;
    bool convex = true;
    for (size_t i = 0; i < size - 1; ++i) {
        Point a = P[(i + 1) % size] - P[i];
        Point b = P[(i + 2) % size] - P[(i + 1) % size];
        if (a * b > eps) {
            convex = false;
        }
    }
    if (!convex) {
        return -1.0;
    }
    double area = 0.0;
    for (size_t i = 0; i < size; ++i) {
        area += dist(P[i], P[(i + 1) % size]);
    }
    if (area < 0.0)area = -1.0 * area;
    return area;
}
```

```
size_t Hexagon::VertexesNumber() {
    return size;
}

Hexagon &Hexagon::operator=(const Hexagon &other) {
    for (size_t i = 0; i < size; ++i) {
        this->P[i] = other.P[i];
    }
    return *this;
}

bool operator==(const Hexagon &a, const Hexagon &b) {
    for (size_t i = 0; i < Hexagon::size; ++i) {
        if (!(a.P[i] == b.P[i]))return false;
    }
    return true;
}

std::ostream &operator<<(std::ostream &os, const Hexagon &hexagon) {
    os << "Hexagon: ";
    for (size_t i = 0; i < Hexagon::size; ++i) {
        os << hexagon.P[i];
    }
    os << "\n";
    return os;
}

std::istream &operator>>(std::istream &is, Hexagon &hexagon) {
    for (size_t i = 0; i < hexagon.size; ++i) {
        is >> hexagon.P[i];
    }
    return is;
}

Hexagon::~Hexagon() {}
```

point.h

```
#ifndef LAB2_POINT_H
#define LAB2_POINT_H

#include <iostream>

class Point {
private:
    double _x, _y;
public:
    Point();

    Point(double x, double y);

    friend bool operator==(const Point &a, const Point &b);

    friend double operator*(const Point &a, const Point &b);

    friend const Point &operator-(const Point &a, const Point &b);

    friend double dist(const Point &a, const Point &b);
```

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

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


#endif //LAB2_POINT_H
```

## point.cpp

```
#include "point.h"

Point::Point() : _x(0.0), _y(0.0) {}

Point::Point(double x, double y) : _x(x), _y(y) {}

double operator*(const Point &a, const Point &b) {
    return a._x * b._y - b._x * a._y;
}

const Point &operator-(const Point &a, const Point &b) {
    Point c;
    c._x = b._x - a._x;
    c._y = b._y - a._y;
    return c;
}

double dist(const Point &a, const Point &b) {
    double dx = (b._x - a._x);
    double mid = (b._y + a._y) / 2.0;
    return dx * mid;
}

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;
}

bool operator==(const Point &a, const Point &b) {
    return (a._x == b._x && a._y == b._y);
}
```

## tlinkedlist_item.h

```
#ifndef LAB2_TLINKEDLIST_ITEM_H
#define LAB2_TLINKEDLIST_ITEM_H

#include "hexagon.h"
#include "iostream"

class TLinkedListItem {
private:
    Hexagon val;
```

```
    TLinkedListItem *next;
public:
    TLinkedListItem(const Hexagon &hexagon, TLinkedListItem *nxt);

    void SetNext(TLinkedListItem *nxt);

    TLinkedListItem *GetNext();

    const Hexagon &GetVal();

    friend std::ostream &operator<<(std::ostream &os,const TLinkedListItem &item);

    virtual ~TLinkedListItem();
};


#endif //LAB2_TLINKEDLIST_ITEM_H
```

## tlinkedlist_item.cpp

```
#include "tlinkedlist_item.h"


TLinkedListItem::~TLinkedListItem() {
    printf("DELETED ITEM\n");
}

TLinkedListItem::TLinkedListItem(const Hexagon &hexagon, TLinkedListItem *nxt) {
    val = hexagon;
    next = nxt;
}

TLinkedListItem *TLinkedListItem::GetNext() {
    return next;
}

void TLinkedListItem::SetNext(TLinkedListItem *nxt) {
    next = nxt;
}

const Hexagon &TLinkedListItem::GetVal() {
    return val;
}

std::ostream &operator<<(std::ostream &os, const TLinkedListItem &item) {
    os << item.val;
    return os;
}
```

## tlinkedlist.h

```
#ifndef LAB2_TLINKEDLIST_H
#define LAB2_TLINKEDLIST_H

#include "hexagon.h"
#include "tlinkedlist_item.h"
#include "iostream"
```

```
class TLinkedList {
private:
    size_t len;
    TLinkedListItem *head;
public:
    TLinkedList();

    TLinkedList(const TLinkedList &list);

    const Hexagon &First();

    const Hexagon &Last();

    void InsertFirst(const Hexagon &hexagon);

    void InsertLast(const Hexagon &hexagon);

    void Insert(const Hexagon &hexagon, size_t pos);

    void RemoveFirst();

    void RemoveLast();

    void Remove(size_t pos);

    const Hexagon &GetItem(size_t ind);

    bool Empty();

    size_t Length();

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

    void Clear();

    virtual ~TLinkedList();
};


#endif //LAB2_TLINKEDLIST_H
```

## tlinkedlist.cpp

```
#include "tlinkedlist.h"

TLinkedList::TLinkedList() {
    len = 0;
    head = nullptr;
}

TLinkedList::TLinkedList(const TLinkedList &list) {
    len = list.len;
    head = new TLinkedListItem(list.head->GetVal(), nullptr);
    TLinkedListItem *cur = head;
    TLinkedListItem *it = list.head;
    for (size_t i = 0; i < len - 1; ++i) {
        it = it->GetNext();
        TLinkedListItem *new_item = new TLinkedListItem(it->GetVal(), nullptr);
        cur->SetNext(new_item);
        cur = cur->GetNext();
```

```
    }
}


const Hexagon &TLinkedList::First() {
    return head->GetVal();
}

const Hexagon &TLinkedList::Last() {
    TLinkedListItem *cur = head;
    for (size_t i = 0; i < len - 1; ++i) {
        cur = cur->GetNext();
    }
    return cur->GetVal();
}

void TLinkedList::InsertFirst(const Hexagon &hexagon) {
    TLinkedListItem *it = new TLinkedListItem(hexagon, head);
    head = it;
    len++;
}

void TLinkedList::InsertLast(const Hexagon &hexagon) {
    TLinkedListItem *cur = head;
    for (size_t i = 0; i < len - 1; ++i) {
        cur = cur->GetNext();
    }
    TLinkedListItem *it = new TLinkedListItem(hexagon, nullptr);
    cur->SetNext(it);
    len++;
}

void TLinkedList::Insert(const Hexagon &hexagon, size_t pos) {
    TLinkedListItem *cur = head;
    TLinkedListItem *prev = nullptr;
    for (size_t i = 0; i < pos; ++i) {
        prev = cur;
        cur = cur->GetNext();
    }
    TLinkedListItem *it = new TLinkedListItem(hexagon, cur);
    if (prev) {
        prev->SetNext(it);
    } else {
        head = it;
    }
    len++;
}

void TLinkedList::RemoveFirst() {
    if (!len)return;
    TLinkedListItem *del = head;
    head = head->GetNext();
    delete del;
    len--;
}

void TLinkedList::RemoveLast() {
    if (!len)return;
    if (len == 1) {
        head = nullptr;
        len = 0;
        return;
```

```
    }
    TLinkedListItem *cur = head;
    for (size_t i = 0; i < len - 2; ++i) {
        cur = cur->GetNext();
    }
    TLinkedListItem *del = cur->GetNext();
    cur->SetNext(nullptr);
    delete del;
    len--;
}

void TLinkedList::Remove(size_t pos) {
    if (!len)return;
    TLinkedListItem *cur = head;
    TLinkedListItem *prev = nullptr;
    for (size_t i = 0; i < pos; ++i) {
        prev = cur;
        cur = cur->GetNext();
    }
    if (prev) {
        prev->SetNext(cur->GetNext());
    } else {
        head = cur->GetNext();
    }
    delete cur;
    len--;
}

const Hexagon &TLinkedList::GetItem(size_t ind) {
    TLinkedListItem *cur = head;
    for (size_t i = 0; i < ind; ++i) {
        cur = cur->GetNext();
    }
    return cur->GetVal();
}

bool TLinkedList::Empty() {
    return len == 0;
}

size_t TLinkedList::Length() {
    return len;
}

std::ostream &operator<<(std::ostream &os, const TLinkedList &list) {
    TLinkedListItem *cur = list.head;
    os << "List: \n";
    for (size_t i = 0; i < list.len; ++i) {
        os << *cur;
        cur = cur->GetNext();
    }
    return os;
}

void TLinkedList::Clear() {
    while (!(this->Empty())) {
        this->RemoveFirst();
    }
}

TLinkedList::~TLinkedList() {
    while (!(this->Empty())) {
```

```
        this->RemoveFirst();
    }
    printf("DELETED LIST\n");
}
```