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

ЛАБОРАТОРНАЯ РАБОТА №6

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:

- Знакомство с шаблонами классов;
- Построение шаблонов динамических структур данных.

Задание

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

- Требования к классам фигуры аналогичны требованиям из лабораторной работы №1;
- Требования к классу контейнера аналогичны требованиям из лабораторной работы №2;
- Шаблон класса-контейнера должен содержать объекты используя std::shared_ptr<…>.

Нельзя использовать:

- Стандартные контейнеры std.

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер.
- Распечатывать содержимое контейнера.
- Удалять фигуры из контейнера.

Описание программы

Исходный код лежит в 9 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\lab4\cmake-build-debug\lab4.exe

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

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

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

2 3 4 5 1 2 3 4 5 1 10 12

List:

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

Hexagon: (2, 3) (4, 5) (1, 2) (3, 4) (5, 1) (10, 12)

List:

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

Hexagon: (2, 3) (4, 5) (1, 2) (3, 4) (5, 1) (10, 12)

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

DELETED ITEM

List:

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

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

DELETED ITEM

DELETED ITEM

DELETED LIST

Process finished with exit code 0

Недочёты

По моему мнению недочетов нет.

Выводы

В данной лабораторной работе я познакомился с шаблонами в языке C++. Я понял, что использование шаблонов при создании структур данных упрощает использования структур для самых различных объектов.

Исходный код

main.cpp

```cpp
#include "tlinkedlist.h"

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

int main() {
    TLinkedList<Hexagon> list;
    shared_ptr<Hexagon> hex1 = make_shared<Hexagon>();
    cin >> *hex1;
    list.InsertLast(hex1);
    cout << *(list.First()) << "\n";
    cout << *(list.Last()) << "\n";
    shared_ptr<Hexagon> hex2 = make_shared<Hexagon>();
    cin >> *hex2;
    list.InsertLast(hex2);
    cout << list << "\n";
    list.Insert(hex1, 2);
    cout << list << "\n";
    list.Remove(1);
    cout << list << "\n";
    return 0;
}
```

hexagon.h

```cpp
#ifndef LAB4_HEXAGON_H
#define LAB4_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 //LAB4_HEXAGON_H
```

hexagon.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 LAB4_POINT_H
#define LAB4_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 //LAB4_POINT_H
```

point.cpp

```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

```cpp
#ifndef LAB4_TLINKEDLIST_ITEM_H
#define LAB4_TLINKEDLIST_ITEM_H

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

using std::shared_ptr;
using std::make_shared;

template <typename T>
class TLinkedListItem {
private:
    shared_ptr<T> val;
    shared_ptr<TLinkedListItem<T>> next;
public:
    TLinkedListItem(shared_ptr<T> hexagon, shared_ptr<TLinkedListItem<T>> nxt);
```

```
    void SetNext(shared_ptr<TLinkedListItem<T>> nxt);

    shared_ptr<TLinkedListItem<T>> GetNext();

    shared_ptr<T> GetVal();

    template<typename X>
    friend std::ostream &operator<<(std::ostream &os, const TLinkedListItem<X>
&item);

    virtual ~TLinkedListItem();
};


#endif //LAB4_TLINKEDLIST_ITEM_H
```

tlinkedlist_item.cpp

```
#include "tlinkedlist_item.h"

template<typename T>
TLinkedListItem<T>::~TLinkedListItem() {
    printf("DELETED ITEM\n");
}

template<typename T>
TLinkedListItem<T>::TLinkedListItem(shared_ptr<T> figure,
shared_ptr<TLinkedListItem<T>> nxt) {
    val = figure;
    next = nxt;
}

template<typename T>
shared_ptr<TLinkedListItem<T>> TLinkedListItem<T>::GetNext() {
    return next;
}

template<typename T>
void TLinkedListItem<T>::SetNext(shared_ptr<TLinkedListItem<T>> nxt) {
    next = nxt;
}

template<typename T>
shared_ptr<T> TLinkedListItem<T>::GetVal() {
    return val;
}

template<typename T>
std::ostream &operator<<(std::ostream &os, const TLinkedListItem<T> &item) {
    os << *item.val;
    return os;
}

template class TLinkedListItem<Hexagon>;
template std::ostream &operator<<(std::ostream &os, const TLinkedListItem<Hexagon>
&item) ;
```

tlinkedlist.h

```h
#ifndef LAB4_TLINKEDLIST_H
#define LAB4_TLINKEDLIST_H

#include "tlinkedlist_item.h"

template<typename T>
class TLinkedList {
private:
    size_t len;
    shared_ptr<TLinkedListItem<T>> head;
public:
    TLinkedList();

    TLinkedList(const TLinkedList<T> &list);

    shared_ptr<T> First();

    shared_ptr<T> Last();

    void InsertFirst(shared_ptr<T> hexagon);

    void InsertLast(shared_ptr<T> hexagon);

    void Insert(shared_ptr<T> hexagon, size_t pos);

    void RemoveFirst();

    void RemoveLast();

    void Remove(size_t pos);

    shared_ptr<T> GetItem(size_t ind);

    bool Empty();

    size_t Length();

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

    void Clear();

    virtual ~TLinkedList();
};


#endif //LAB4_TLINKEDLIST_H
```

tlinkedlist.cpp

```cpp
#include "tlinkedlist.h"

template<typename T>
TLinkedList<T>::TLinkedList() {
    len = 0;
    head = nullptr;
}

template<typename T>
TLinkedList<T>::TLinkedList(const TLinkedList<T> &list) {
```

```
    len = list.len;
    if(!list.len){
        head = nullptr;
        return;
    }
    head = make_shared<TLinkedListItem<T>>(list.head->GetVal(), nullptr);
    shared_ptr<TLinkedListItem<T>> cur = head;
    shared_ptr<TLinkedListItem<T>> it = list.head;
    for (size_t i = 0; i < len - 1; ++i) {
        it = it->GetNext();
        shared_ptr<TLinkedListItem<T>> new_item = make_shared<TLinkedListItem<T>>(it-
>GetVal(), nullptr);
        cur->SetNext(new_item);
        cur = cur->GetNext();
    }
}

template<typename T>
shared_ptr<T> TLinkedList<T>::First() {
    if (len == 0) {
        return nullptr;
    }
    return head->GetVal();
}

template<typename T>
shared_ptr<T> TLinkedList<T>::Last() {
    if (len == 0) {
        return nullptr;
    }
    shared_ptr<TLinkedListItem<T>> cur = head;
    for (size_t i = 0; i < len - 1; ++i) {
        cur = cur->GetNext();
    }
    return cur->GetVal();
}

template<typename T>
void TLinkedList<T>::InsertFirst(shared_ptr<T> figure) {
    shared_ptr<TLinkedListItem<T>> it = make_shared<TLinkedListItem<T>>(figure,
head);
    head = it;
    len++;
}

template<typename T>
void TLinkedList<T>::InsertLast(shared_ptr<T> figure) {
    if (len == 0) {
        head = make_shared<TLinkedListItem<T>>(figure, nullptr);
        len = 1;
        return;
    }
    shared_ptr<TLinkedListItem<T>> cur = head;
    for (size_t i = 0; i < len - 1; ++i) {
        cur = cur->GetNext();
    }
    shared_ptr<TLinkedListItem<T>> it = make_shared<TLinkedListItem<T>>(figure,
nullptr);
    cur->SetNext(it);
    len++;
}
```

```cpp
template<typename T>
void TLinkedList<T>::Insert(shared_ptr<T> figure, size_t pos) {
    if (pos > len || pos < 0) {
        return;
    }
    shared_ptr<TLinkedListItem<T>> cur = head;
    shared_ptr<TLinkedListItem<T>> prev = nullptr;
    for (size_t i = 0; i < pos; ++i) {
        prev = cur;
        cur = cur->GetNext();
    }
    shared_ptr<TLinkedListItem<T>> it = make_shared<TLinkedListItem<T>>(figure, cur);
    if (prev) {
        prev->SetNext(it);
    } else {
        head = it;
    }
    len++;
}

template<typename T>
void TLinkedList<T>::RemoveFirst() {
    if (!len)return;
    shared_ptr<TLinkedListItem<T>> del = head;
    head = head->GetNext();
    len--;
}

template<typename T>
void TLinkedList<T>::RemoveLast() {
    if (!len)return;
    if (len == 1) {
        head = nullptr;
        len = 0;
        return;
    }
    shared_ptr<TLinkedListItem<T>> cur = head;
    for (size_t i = 0; i < len - 2; ++i) {
        cur = cur->GetNext();
    }
    shared_ptr<TLinkedListItem<T>> del = cur->GetNext();
    cur->SetNext(nullptr);
    len--;
}

template<typename T>
void TLinkedList<T>::Remove(size_t pos) {
    if (!len)return;
    if (pos < 0 || pos >= len)return;
    shared_ptr<TLinkedListItem<T>> cur = head;
    shared_ptr<TLinkedListItem<T>> 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();
    }
    len--;
}
```

```cpp
template<typename T>
shared_ptr<T> TLinkedList<T>::GetItem(size_t ind) {
    if (ind < 0 || ind >= len)return nullptr;
    shared_ptr<TLinkedListItem<T>> cur = head;
    for (size_t i = 0; i < ind; ++i) {
        cur = cur->GetNext();
    }
    return cur->GetVal();
}

template<typename T>
bool TLinkedList<T>::Empty() {
    return len == 0;
}

template<typename T>
size_t TLinkedList<T>::Length() {
    return len;
}

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

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

template<typename T>
TLinkedList<T>::~TLinkedList() {
    while (!(this->Empty())) {
        this->RemoveFirst();
    }
    printf("DELETED LIST\n");
}

template
class TLinkedList<Hexagon>;

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