

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

ЛАБОРАТОРНАЯ РАБОТА №2

по курсу “Объектно-ориентированное программирование» 1 семестр,
2021/22 уч. год

Студент: Колпакова Диана Саргаевна, группа М8О-208Б-20

Преподаватель: Дорохов Евгений Павлович

Задание

1. Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **одну фигуру (колонка фигура 1)**, согласно вариантам задания. Классы должны удовлетворять следующим правилам:
 - Требования к классу фигуры аналогичны требованиям из лабораторной работы No1.
 - 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	Метод по получению	Метод по удалению
контейнер фигуры из контейнера	фигуры из контейнера	фигуры из контейнера
Очередь: Push Динамический массив: InsertLast Связанный список: InsertFirst, InsertLast, Insert Бинарное дерево: Push N-дерево: Update		
Очередь: Top Динамический массив: operator[] Связанный список: First, Last, GetElement Бинарное дерево: GetNotLess N-дерево: GetItem		
Очередь: Pop Динамический массив: Remove Связанный список: RemoveFirst, RemoveLast, Remove Бинарное дерево: Pop N-дерево: RemoveSubTree <ul style="list-style-type: none">○ Перегруженный оператор по выводу контейнера в поток <code>std::ostream (<<)</code>;○ Деструктор, удаляющий все элементы контейнера;○ Набор специальных методов для класса-контейнера (см. Приложение).		

Полное описание всех методов можно найти в приложении к лабораторной. Нельзя использовать:

- Стандартные контейнеры `std`;
- Шаблоны (`template`);
- Различные варианты умных указателей (`unique_ptr`, `shared_ptr`, `weak_ptr`,...).

Программа должна позволять:

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

Вариант 9:

Фигура №1	Имя класса	Фигура №2	Имя класса	Фигура №3	Имя класса
Треугольник	Triangle	Квадрат	Square	Прямоугольник	Rectangle

Описание программы

Исходный код лежит в 9 файлах:

1. main.cpp: часть программы, отвечающая за взаимодействие с пользователем через консоль. В ней происходит инициализация объектов и вызов функций работы с ними, заполнение стандартного контейнера вектор введенными объектами и печать его содержимого;
2. point.h: описание класса Point точек A(a1, a2);
3. point.cpp: реализация класса Point;
4. figure.h: описание абстрактного класса-родителя Figure;
5. figure.cpp: реализация класса Figure;
6. triangle.h: описание класса Triangle треугольников, заданных по трем точкам, наследника Figure;
7. triangle.cpp: реализация класса Triangle;
8. TLinkedList.cpp: реализация класса связного списка
9. TLinkedList.h: описание класса связного списка

Также используется файл CMakeLists.txt с конфигурацией CMake для автоматизации сборки программы.

Дневник отладки

Проблем не возникло

Вывод

В процессе выполнения лабораторной я на практике познакомилась с работой

класса-контейнера связный список. Реализовала его, его функции и контейнеры, выполнила перегрузку оператора вывода. Рада, что пока не столкнулась с необходимостью выделения памяти с помощью команд new и delete.

Исходный код

main.cpp:

```
// OOP, Lab 2 variant 9, Diana Kolpakova
// Triangle, TLinkedList

#include <iostream>

#include "figure.h"
#include "triangle.h"
#include "tlinkedlist.h"

using namespace std;

int main()
{
    cout.setf(ios_base::boolalpha);
    cout << "oop_exercise_2 (c) Diana Kolpakova" << endl;
    cout << "Triangles, TLinkedList" << endl;

    TLinkedList list = TLinkedList();

    for (;;)
    {
        cout << endl;
        cout << "Select an action for the linked list of triangles" << endl;
        cout << "1) Is the list empty?" << endl;
        cout << "2) Get number of triangles in the list" << endl;
        cout << "3) Show the first triangle from the list" << endl;
        cout << "4) Show the last triangle from the list" << endl;
        cout << "5) Show the triangle at a specified position in the list" <<
endl;
        cout << "6) Show areas of all triangles in the list" << endl;
        cout << "7) Add a new triangle to the beginning of the list" << endl;
        cout << "8) Add a new triangle to the end of the list" << endl;
        cout << "9) Add a new triangle to a specified position in the list" <<
endl;
        cout << "a) Remove the new first triangle from the list" << endl;
        cout << "b) Remove the new last triangle from the list" << endl;
        cout << "c) Remove the triangle at a specified position in the list" <<
endl;
        cout << "d) Remove all triangles from the list" << endl;
        cout << "x) End the program" << endl;

        try
        {
            Triangle triangle;
            size_t position;
```

```

char ch;
cin >> ch;
switch (ch)
{
    case '1':
        cout << "Is the list empty: " << list.Empty() << endl;
        break;
    case '2':
        cout << "Length of the list: " << list.Length() << endl;
        break;
    case '3':
        triangle = list.First();
        cout << triangle << endl;
        break;
    case '4':
        triangle = list.Last();
        cout << triangle << endl;
        break;
    case '5':
        cout << "Enter position in the list:";
        cin >> position;
        triangle = list.GetItem(position);
        cout << triangle << endl;
        break;
    case '6':
        cout << "Triangle areas:" << endl;
        if (list.Empty())
        {
            cout << "Empty list" << endl;
        }
        else
        {
            cout << list << endl;
        }
        break;
    case '7':
        cout << "Enter 3 points of triangle (6 numbers):";
        cin >> triangle;
        list.InsertFirst(triangle);
        cout << triangle << endl;
        break;
    case '8':
        cout << "Enter 3 points of triangle (6 numbers):";
        cin >> triangle;
        list.InsertLast(triangle);
        cout << triangle << endl;
        break;
    case '9':
        cout << "Enter 3 points of triangle (6 numbers):";
        cin >> triangle;
        cout << "Enter position in the list:";
        cin >> position;
        list.Insert(triangle, position);
        cout << triangle << endl;
        break;
    case 'a':
    case 'A':

```

```

        list.RemoveFirst();
        cout << "Removed the first triangle" << endl;
        break;
    case 'b':
    case 'B':
        list.RemoveLast();
        cout << "Removed the last triangle" << endl;
        break;
    case 'c':
    case 'C':
        cout << "Enter position in the list:";
        cin >> position;
        list.Remove(position);
        cout << "Removed the triangle at specified position" <<
endl;
        break;
    case 'd':
    case 'D':
        list.Clear();
        cout << "Removed all" << endl;
        break;
    case 'l':
    case 'L':
        cout << "Triangles:" << endl;
        if (list.Empty())
        {
            cout << "Empty list" << endl;
        }
        else
        {
            for (size_t i = 0; i < list.Length(); i++)
            {
                triangle = list.GetItem(i);
                cout << "#" << i << " " << triangle << endl;
            }
        }
        break;
    case 'q':
    case 'Q':
    case 'x':
    case 'X':
        cout << "Exiting" << endl;
        return 0;
    default:
        cout << "Error: invalid action selected" << endl;
        break;
    }
}
catch (exception& ex)
{
    cout << "Exception: " << ex.what() << endl;
}
}
}

```

point.h:

```
#pragma once
#include <iostream>

using namespace std;

class Point
{
private:
    double x;
    double y;

public:
    Point();
    Point(double x, double y);

    static double Distance (const Point& point1, const Point& point2);

    friend istream& operator>>(istream& is, Point& point);
    friend ostream& operator<<(ostream& os, Point& point);

    bool operator==(const Point& other);
};
```

point.cpp:

```
#include <cmath>
#include "point.h"

using namespace std;

Point::Point()
{
    this->x = 0.0;
    this->y = 0.0;
}

Point::Point(double x, double y)
{
    this->x = x;
    this->y = y;
}

double Point::Distance(const Point& point1, const Point& point2)
{
    double dx = point1.x - point2.y;
    double dy = point1.y - point2.y;
    double distance = sqrt(dx * dx + dy * dy);
    return distance;
}

bool Point::operator==(const Point& other)
{
}
```

```

        return (this->x == other.x)
            && (this->y == other.y);
    }

    istream& operator>>(istream& is, Point& point)
    {
        is >> point.x >> point.y;
        return is;
    }

    ostream& operator<<(ostream& os, Point& point)
    {
        os << "(" << point.x << ", " << point.y << ")";
        return os;
    }

```

figure.h:

```

#pragma once
#include "point.h"

class Figure
{
public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
};

```

triangle.h:

```

#pragma once
#include "figure.h"

class Triangle : public Figure
{
private:
    Point point1;
    Point point2;
    Point point3;

public:
    Triangle();
    Triangle(Point point1, Point point2, Point point3);
    Triangle(const Triangle& other);

    virtual size_t VertexesNumber() override;
    virtual double Area() override;

    friend istream& operator>>(istream& is, Triangle& triangle);
    friend ostream& operator<<(ostream& os, Triangle& triangle);

    Triangle& operator=(const Triangle& other);

```



```
    bool operator==(const Triangle& other);  
};
```

triangle.cpp:

```
#include "triangle.h"  
  
using namespace std;  
  
Triangle::Triangle()  
{  
    this->point1 = Point();  
    this->point2 = Point();  
    this->point3 = Point();  
}  
  
Triangle::Triangle(Point point1, Point point2, Point point3)  
{  
    this->point1 = point1;  
    this->point2 = point2;  
    this->point3 = point3;  
}  
  
Triangle::Triangle(const Triangle& other)  
{  
    this->point1 = other.point1;  
    this->point2 = other.point2;  
    this->point3 = other.point3;  
}  
  
size_t Triangle::VertexesNumber()  
{  
    return 3;  
}  
  
double Triangle::Area()  
{  
    double length12 = Point::Distance(point1, point2);  
    double length23 = Point::Distance(point2, point3);  
    double length31 = Point::Distance(point3, point1);  
    double semiPerimeter = (length12 + length23 + length31) / 2.0;  
    return sqrt(semiPerimeter * (semiPerimeter - length12) * (semiPerimeter -  
length23) * (semiPerimeter - length31));  
}  
  
istream& operator>>(istream& is, Triangle& triangle)  
{  
    is >> triangle.point1 >> triangle.point2 >> triangle.point3;  
    return is;  
}  
  
ostream& operator<<(ostream& os, Triangle& triangle)  
{  
    os << "Triangle: " << triangle.point1 << ", " << triangle.point2 << ", " <<  
triangle.point3;  
    return os;  
}
```

```

Triangle& Triangle::operator=(const Triangle& other)
{
    this->point1 = other.point1;
    this->point2 = other.point2;
    this->point3 = other.point3;
    return *this;
}

bool Triangle::operator==(const Triangle& other)
{
    return (this->point1 == other.point1)
        && (this->point2 == other.point2)
        && (this->point3 == other.point3);
}

```

TLinkedList.h:

```

#pragma once
#include "triangle.h"

class TLinkedList
{
private:
    struct Item
    {
        Triangle triangle;
        Item* pNextItem{};
    };

    size_t length;
    Item* pFirstItem;
    Item* pLastItem;

public:
    TLinkedList();
    TLinkedList(const TLinkedList& other);
    virtual ~TLinkedList();

    const Triangle& First();
    const Triangle& Last();
    const Triangle& GetItem(size_t position);

    void InsertFirst(const Triangle& triangle);
    void InsertLast(const Triangle& triangle);
    void Insert(const Triangle& triangle, size_t position);

    void RemoveFirst();
    void RemoveLast();
    void Remove(size_t position);

    void Clear();
    bool Empty();
    size_t Length();

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

```

TLinkedList.cpp:

```
#include "tlinkedlist.h"

TLinkedList::TLinkedList()
{
    pFirstItem = nullptr;
    pLastItem = nullptr;
    length = 0;
}

TLinkedList::TLinkedList(const TLinkedList& other)
{
    pFirstItem = nullptr;
    pLastItem = nullptr;
    length = 0;

    Item* pCurrentItem = other.pFirstItem;
    while (pCurrentItem != nullptr)
    {
        InsertLast(pCurrentItem->triangle);
        pCurrentItem = pCurrentItem->pNextItem;
    }
}

const Triangle& TLinkedList::First()
{
    if (Empty())
        throw runtime_error("Cannot get the item from empty list");
    return pFirstItem->triangle;
}

const Triangle& TLinkedList::Last()
{
    if (Empty())
        throw runtime_error("Cannot get the item from empty list");
    return pLastItem->triangle;
}

void TLinkedList::InsertFirst(const Triangle& triangle)
{
    Item* pNewItem = new Item();
    pNewItem->triangle = triangle;
    pNewItem->pNextItem = pFirstItem;

    pFirstItem = pNewItem;
    if (Empty())
        pLastItem = pNewItem;

    length++;
}

void TLinkedList::InsertLast(const Triangle& triangle)
{

```

```

    Item* pNewItem = new Item();
    pNewItem->triangle = triangle;
    pNewItem->pNextItem = nullptr;

    if (pLastItem != nullptr)
        pLastItem->pNextItem = pNewItem;
    pLastItem = pNewItem;
    if (Empty())
        pFirstItem = pNewItem;

    length++;
}

void TLinkedList::Insert(const Triangle& triangle, size_t position)
{
    if (position == 0)
    {
        InsertFirst(triangle);
        return;
    }
    else if (position == length)
    {
        InsertLast(triangle);
        return;
    }
    else if (position > length)
        throw runtime_error("Specified poition is out of range");

    int i = 0;
    Item* pCurrentItem = pFirstItem;
    Item* pPreviousItem = nullptr;
    while (pCurrentItem != nullptr)
    {
        if (i == position)
            break;
        pPreviousItem = pCurrentItem;
        pCurrentItem = pCurrentItem->pNextItem;
        i++;
    }

    Item* pNewItem = new Item();
    pNewItem->triangle = triangle;
    pNewItem->pNextItem = pCurrentItem;

    pPreviousItem->pNextItem = pNewItem;

    length++;
}

void TLinkedList::RemoveFirst()
{
    if (Empty())
        throw runtime_error("Cannon remove the item from empty list");
    Item* pNextItem = pFirstItem->pNextItem;
    delete pFirstItem;
    pFirstItem = pNextItem;
}

```

```

    length--;
    if (Empty())
        pLastItem = nullptr;
}

void TLinkedList::RemoveLast()
{
    if (Empty())
        throw runtime_error("Cannon remove the item from empty list");

    Item* pCurrentItem = pFirstItem;
    Item* pPreviousItem = nullptr;
    while (pCurrentItem != nullptr)
    {
        if (pCurrentItem == pLastItem)
            break;
        pPreviousItem = pCurrentItem;
        pCurrentItem = pCurrentItem->pNextItem;
    }

    if (pPreviousItem != nullptr)
        pPreviousItem->pNextItem = nullptr;
    delete pLastItem;
    pLastItem = pPreviousItem;
    length--;
    if (Empty())
        pFirstItem = nullptr;
}

void TLinkedList::Remove(size_t position)
{
    if (Empty())
        throw runtime_error("Cannon remove the item from empty list");
    if (position == 0)
    {
        RemoveFirst();
        return;
    }
    else if (position == length - 1)
    {
        RemoveLast();
        return;
    }
    else if (position >= length)
        throw runtime_error("Specified poition is out of range");

    int i = 0;
    Item* pCurrentItem = pFirstItem;
    Item* pPreviousItem = nullptr;
    while (pCurrentItem != nullptr)
    {
        if (i == position)
            break;
        pPreviousItem = pCurrentItem;
        pCurrentItem = pCurrentItem->pNextItem;
        i++;
    }
}

```

```

    pPreviousItem->pNextItem = pCurrentItem->pNextItem;
    delete pCurrentItem;
    length--;
}

const Triangle& TLinkedList::GetItem(size_t position)
{
    if (Empty())
        throw runtime_error("Cannot get the item from empty list");
    if (position >= length)
        throw runtime_error("Specified position is out of range");

    int i = 0;
    Item* pCurrentItem = pFirstItem;
    while (pCurrentItem != nullptr)
    {
        if (i == position)
            return pCurrentItem->triangle;
        pCurrentItem = pCurrentItem->pNextItem;
        i++;
    }

    throw runtime_error("Something went wrong");
}

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

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

void TLinkedList::Clear()
{
    Item* pCurrentItem = pFirstItem;
    while (pCurrentItem != nullptr)
    {
        Item* pNextItem = pCurrentItem->pNextItem;
        delete pCurrentItem;
        pCurrentItem = pNextItem;
    }
    pFirstItem = nullptr;
    pLastItem = nullptr;
    length = 0;
}

TLinkedList::~~TLinkedList()
{
    Clear();
}

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

```

```

TLinkedList::Item* pCurrentItem = list.pFirstItem;
while (pCurrentItem != nullptr)
{
    os << pCurrentItem->triangle.Area();
    if (pCurrentItem != list.pLastItem)
        os << " -> ";
    pCurrentItem = pCurrentItem->pNextItem;
}
return os;
}

```

CMakeLists.txt:

```

cmake_minimum_required(VERSION 3.21)
project(oop_exercise_2)

set(CMAKE_CXX_STANDARD 14)

include_directories(.)

add_executable(oop_exercise_2
    figure.cpp
    figure.h
    main.cpp
    point.cpp
    point.h
    tlinkedlist.cpp
    tlinkedlist.h
    triangle.cpp
    triangle.h)

```