МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №4**

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

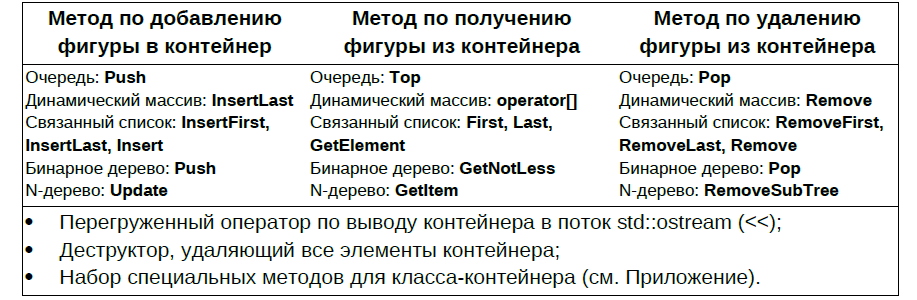
Студентка: *Волошинская Евгения Владимировна, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович*

**Задание**

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 №1;
* Иметь общий родительский класс Figure;
* 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:



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** | **Контейнер 1-го уровня** | **Имя класса** |
| Треугольник | Triangle | Связанный список | TLinkedList |

**Вариант: Связный список:**

class TLinkedList {

public:

// Конструктор по умолчанию

TLinkedList();

// Конструктор копирования

TLinkedList(const TLinkedList& other);

// Метод, возвращающий первую фигуру списка

const Polygon& First();

// Метод, возвращающий последнюю фигуру списка

const Polygon& Last();

// Метод, добавляющий элемент в начало списка

void InsertFirst(const Polygon& polygon);

// Метод, добавляющий фигуру в конец списка

void InsertLast(const Polygon& polygon);

// Метод, добавляющий фигуру в произвольное место списка

void Insert(const Polygon& polygon, size\_t position);

// Метод, удаляющий первый элемент списка

void RemoveFirst();

// Метод, удаляющий последний элемент списка

void RemoveLast();

// Метод, удаляющий произвольный элемент списка

void Remove(size\_t position);

// Метод получения фигуры списка по индексу.

const Polygon& GetItem(size\_t idx);

// Метод, проверяющий пустоту списка

bool Empty();

// Метод, возвращающий длину массива

size\_t Length();

// Оператор вывода для массива в формате:

// "S1 -> S2 -> ... -> Sn", где Si - площадь фигуры

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

// Метод, удаляющий все элементы контейнера,

// но позволяющий пользоваться им.

void Clear();

// Деструктор

virtual ~TLinkedList();

};

**Описание программы**

Исходный код лежит в 10 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. item.h: описание класса Item, объектами которого являются элементы связанного списка;
9. item.cpp: реализация класса Item;
10. tlinkedlist.h: описание класса TLinkedList, объекты которого – связанные списки элементов типа Item;
11. tlinkedlist.cpp: реализация класса TLinkedList.

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

**Дневник отладки**

***Ошибка***: error: ‘Item\* Item::next’ is private within this context

for (Item\* i = head; i != nullptr; i = i -> next)

***Решение***: добавить функции Right(), Left() в классе Item для получения соседних элементов, чтобы была возможность доступа к закрытым полям-указателям next и prev класса Item в функциях, не принадлежащих классу, через Right(), Left().

***Ошибка***: tlinkedlist.cpp:78:21: error: lvalue required as left operand of assignment

item->Right() = head;

***Решение***: добавить функции InsRight, InsLeft для связи установления соседнего элемента передаваемым.

***Ошибка***: tlinkedlist.cpp: In function ‘void InsertFirst(const Triangle&)’:

error: ‘head’ was not declared in this scope

if (head == nullptr) {

***Решение***: добавить TLinkedList:: перед названием функции над связным списком.

***Ошибка***: triangle.cpp: In member function ‘bool Triangle::operator==(const Triangle&)’:

triangle.cpp:38:13: error: no match for ‘operator==’ (operand types are ‘Point’ and ‘const Point’)

if ((p1 == other.p1) && (p2 == other.p2) && (p3 == other.p3))

***Решение***: добавить перегрузку оператора == для класса Point.

**Вывод**

В данной лабораторной работе я продолжила знакомиться с основами ООП в языке C++. Я получила навык создания своего контейнера в C++ с нуля, в моем случае это был связный список. В качестве элементов контейнера были приняты объекты класса Triangle из предыдущей лабораторной работы. Для того, чтобы реализовать связь между элементами контейнера, был создан вспомогательный класс Item, экземпляры которого содержат элемент и ссылки на соседние элементы. Были запрограммированы функции получения первого и последнего элементов списка, измерения длины, очищения списка, а также вставки и удаления элементов списка. В результате, я получила опыт создания контейнеров и работы с ними в языке C++, а также закрепила уже имеющиеся знания.

**Исходный код**

main.cpp:

#include "tlinkedlist.h"

int main(void)

{

TLinkedList l;

Point a1(-3, -1);

Point b1(3, 0);

Point c1(4, 8);

Point a2(0, 0);

Point b2(2, 3);

Point c2(-2, 6);

Point a3(1, 0);

Point b3(0.5, 1);

Point c3(2, 1);

Triangle t1(a1, b1, c1);

Triangle t2(a2, b2, c2);

Triangle t3(a3, b3, c3);

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

l.Insert(t1, 1);

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

l.Insert(t1, 3);

l.Insert(t2, 2);

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

l.InsertLast(t1);

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

l.Insert(t3, 4);

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

l.Insert(t3, 3);

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

l.Insert(t2, 6);

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

l.Insert(t2, 1);

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

l.InsertFirst(t3);

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

l.Remove(9);

l.Remove(5);

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

std::cout << "Length: " << 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(t3);

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

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

std::cout << l.Last() << 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;

l.Clear();

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

return 0;

}

point.h:

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

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

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

public:

Point();

Point(double x, double y);

Point(std::istream &is);

bool operator==(const Point &other);

double dist(Point& other);

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\_;

}

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

{

return ((x\_ == other.x\_) && (y\_ == other.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, const Point& p) {

os << "(" << p.x\_ << ", " << p.y\_ << ")";

return os;

}

figure.h:

#ifndef FIGURE\_H

#define FIGURE\_H

#include "point.h"

class Figure {

public:

virtual size\_t VertexesNumber() = 0;

virtual void Print(std::ostream& os) = 0;

virtual double Area() = 0;

virtual ~Figure() {};

};

#endif // FIGURE\_H

triangle.h:

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include <iostream>

#include "figure.h"

class Triangle : public Figure {

public:

Triangle();

Triangle(Point a, Point b, Point c);

Triangle(std::istream &is);

Triangle(const Triangle& other);

Triangle &operator=(const Triangle &other);

bool operator==(const Triangle &other);

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

friend std::ostream& operator<<(std::ostream& os, const Triangle& t);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

virtual ~Triangle();

private:

Point p1;

Point p2;

Point p3;

};

#endif // TRIANGLE\_H

triangle.cpp:

#include "triangle.h"

#include <iostream>

#include <cmath>

Triangle::Triangle()

: p1(0.0, 0.0), p2(0.0, 0.0), p3(0.0, 0.0) { // можно, но длиннее p1(Point(0.0, 0.0))

//std::cout << "Default triangle created" << std::endl;

}

Triangle::Triangle(Point a, Point b, Point c)

: p1(a), p2(b), p3(c) {

//std::cout << "Triangle created by parameters" << std::endl;

}

Triangle::Triangle(std::istream &is) {

is >> p1 >> p2 >> p3;

}

Triangle::Triangle(const Triangle& other)

: Triangle(other.p1, other.p2, other.p3) {

//std::cout << "Triangle copy created" << std::endl;

}

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

{

if (this == &other) {

return \*this;

}

p1 = other.p1;

p2 = other.p2;

p3 = other.p3;

return \*this;

}

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

{

return (p1 == other.p1) && (p2 == other.p2) && (p3 == other.p3);

}

std::istream& operator>>(std::istream& is, Triangle& t)

{

is >> t.p1 >> t.p2 >> t.p3;

return is;

}

std::ostream& operator<<(std::ostream& os, const Triangle& t)

{

os << "Triangle: " << t.p1 << " " << t.p2 << " " << t.p3 << std::endl;

return os;

}

size\_t Triangle::VertexesNumber() {

return(size\_t)3;

}

double Triangle::Area() {

double p12 = p1.dist(p2);

double p13 = p1.dist(p3);

double p23 = p2.dist(p3);

double p = (p12 + p23 + p13) / 2.0;

return std::sqrt(p \* (p - p12) \* (p - p23) \* (p - p13));

}

void Triangle::Print(std::ostream& os) {

os << "Triangle: ";

os << p1 << ", ";

os << p2 << ", ";

os << p3 << std::endl;

}

Triangle::~Triangle() {

//std::cout << "Triangle deleted" << std::endl;

}

item.h:

#ifndef ITEM\_H

#define ITEM\_H

#include "triangle.h"

class Item

{

public:

Item(const Triangle &s);

Item(const Item &other);

Item\* Left();

Item\* Right();

void InsLeft(Item\* item);

void InsRight(Item\* item);

Triangle& GetTriangle();

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

virtual ~Item();

private:

Triangle triangle;

Item\* prev;

Item\* next;

};

#endif // ITEM\_H

item.cpp:

#include "item.h"

Item::Item(const Triangle &t)

{

this->triangle = t;

this->next = nullptr;

this->prev = nullptr;

}

Item::Item(const Item &other)

{

this->triangle = other.triangle;

this->next = other.next;

this->prev = other.prev;

}

Item \*Item::Left()

{

return this->prev;

}

Item\* Item::Right()

{

return this->next;

}

void Item::InsLeft(Item\* item)

{

this->prev = item;

}

void Item::InsRight(Item\* item)

{

this->next = item;

}

Triangle& Item::GetTriangle()

{

return this->triangle;

}

std::ostream &operator<<(std::ostream &os, const Item &item)

{

os << item.triangle << std::endl;

return os;

}

Item::~Item() {}

tlinkedlist.h:

#ifndef TLINKEDLIST\_H

#define TLINKEDLIST\_H

#include "item.h"

class TLinkedList

{

public:

TLinkedList();

TLinkedList(const TLinkedList& other);

size\_t Length();

bool Empty();

const Triangle& First();

const Triangle& Last();

const Triangle& GetItem(size\_t idx);

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);

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

void Clear();

virtual ~TLinkedList();

private:

Item\* head;

Item\* tail;

};

#endif // TLINKEDLIST\_H

tlinkedlist.cpp:

#include "tlinkedlist.h"

TLinkedList::TLinkedList() : head(nullptr), tail(nullptr) {}

TLinkedList::TLinkedList(const TLinkedList &other)

{

head = other.head;

tail = other.tail;

}

bool TLinkedList::Empty()

{

return (head == nullptr);

}

size\_t TLinkedList::Length()

{

size\_t size = 0;

for (Item\* i = head; i != nullptr; i = i->Right()) {

++size;

}

return size;

}

const Triangle& TLinkedList::First()

{

if (head == nullptr) {

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

exit(1);

}

return head -> GetTriangle();

}

const Triangle& TLinkedList::Last()

{

if (head == nullptr) {

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

exit(1);

}

Item\* pi = head;

Item\* i = head->Right();

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

pi = i;

}

return pi -> GetTriangle();

}

const Triangle& TLinkedList::GetItem(size\_t idx)

{

size\_t len = Length();

if (head == nullptr) {

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

exit(1);

}

if (idx > len) {

std::cout << "No element on position " << idx << std::endl;

exit(1);

}

Item\* item = head;

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

item = item->Right();

}

return item->GetTriangle();

}

void TLinkedList::InsertFirst(const Triangle& triangle)

{

Item\* item = new Item(triangle);

if (head == nullptr) {

head = item;

tail = item;

return;

} // важно, или будет обращение к nullptr -> prev

//item->InsLeft(nullptr);

item->InsRight(head);

head->InsLeft(item);

head = item;

}

void TLinkedList::InsertLast(const Triangle& triangle)

{

Item\* item = new Item(triangle);

if (head == nullptr) {

head = item;

tail = item;

return;

}

tail->InsRight(item);

item->InsLeft(tail);

//item->InsRight(nullptr);

tail = item;

}

void TLinkedList::Insert(const Triangle& triangle, size\_t position)

{

size\_t len = Length();

if (position > len + 1) {

std::cout << "No such position" << std::endl;

return;

}

if (position == 1) {

InsertFirst(triangle);

return;

}

if (position == len + 1) {

InsertLast(triangle);

return;

}

Item\* item = new Item(triangle);

Item\* curr = head;

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

curr = curr->Right();

}

Item\* prev = curr->Left();

prev->InsRight(item);

curr->InsLeft(item);

item->InsLeft(prev);

item->InsRight(curr);

}

void TLinkedList::RemoveFirst()

{

if (head == nullptr) {

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

return;

}

if (head == tail) {

delete head;

head = nullptr;

tail = nullptr;

return;

}

Item\* item = head;

head = head->Right();

head->InsLeft(nullptr);

delete item;

}

void TLinkedList::RemoveLast()

{

if (head == nullptr) {

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

return;

}

if (head == tail) {

delete head;

head = nullptr;

tail = nullptr;

return;

}

Item\* item = tail;

tail = tail->Left();

tail->InsRight(nullptr);

delete item;

}

void TLinkedList::Remove(size\_t position)

{

size\_t len = Length();

if (head == nullptr) {

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

return;

}

if (position > len) {

std::cout << "No such position" << std::endl;

return;

}

if (position == 1) {

RemoveFirst();

return;

}

if (position == len) {

RemoveLast();

return;

}

Item\* item = head;

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

item = item->Right();

}

Item\* left = item->Left();

Item\* right = item->Right();

left->InsRight(right);

right->InsLeft(left);

delete item;

}

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

{

if (list.head == nullptr) {

os << "List is empty";

return os;

}

for (Item\* i = list.head; i != nullptr; i = i->Right()) {

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

os << i->GetTriangle().Area() << " -> ";

else

os << i->GetTriangle().Area();

}

return os;

}

void TLinkedList::Clear()

{

while (head != nullptr) {

RemoveFirst();

}

}

TLinkedList::~TLinkedList()

{

while (head != nullptr) {

RemoveFirst();

}

}

CMakeLists.txt:

cmake\_minimum\_required(VERSION 3.10)

project(lab2)

set(CMAKE\_CXX\_STANDARD 11)

add\_executable(lab2 point.h

point.cpp

main.cpp

figure.h

triangle.h triangle.cpp

item.h item.cpp tlinkedlist.h tlinkedlist.cpp)