МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №4**

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:** Спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру. Классы должны удовлетворять следующим правилам:

* Требования к классу фигуры аналогичны требованиям из лабораторной работы 2.
* Классы фигур должны содержать набор следующих методов:
  + Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>)
  + Перегруженный оператор вывода в поток std::ostream (<<)
  + 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:
  + Length() – возвращает количество элементов в контейнере
  + Empty() – для пустого контейнера возвращает 1, иначе – 0
  + First() – возвращает первый (левый) элемент списка
  + Last() – возвращает последний (правый) элемент списка
  + InsertFirst(elem) – добавляет элемент в начало списка
  + RemoveFirst() – удаляет элемент из начала списка
  + InsertLast(elem) – добавляет элемент в конец списка
  + RemoveLast() – удаляет элемент из конца списка
  + Insert(elem, pos) – вставляет элемент на позицию pos
  + Remove(pos) – удаляет элемент, находящийся на позиции pos
  + Clear() – удаляет все элементы из списка
  + operator<< – выводит список поэлементно в поток вывода (слева направо)

Нельзя использовать:

* Стандартные контейнеры std.
* Шаблоны (template).
* Различные варианты умных указателей (shared\_ptr, weak\_ptr).

Программа должна позволять:

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

**Вариант №8:**

* + Фигура: Восьмиугольник (Octagon)
  + Контейнер: Список (TLinkedList)

**Описание программы:**

Исходный код разделён на 9 файлов:

* point.h – описание класса точки
* point.cpp – реализация класса точки
* 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:**

В item.cpp не получилось взять элемент по значению, (метод GetOctagon()), в итоге возвращала сслыку на восьмиугольник.

**Вывод:**  
 При выполнении работы я на практике освоила основы работы класса-контейнера, реализовала двусвязный список, конструкторы и функции для работы с ним. Я перегрузила оператор вывода, это оказалось не сложнее, чем в предыдущей лабораторной, только выводятся не восьмиугольники, а их площади. Также я вспомнила с 1-го курса, как на базовом уровне работать с выделением и очисткой памяти на языке C++ при помощи команд new и delete.

**Исходный код:**

**figure.h**

1. #ifndef FIGURE\_H
2. #define FIGURE\_H
4. #include "point.h"
6. class Figure {
7. public:
8. virtual size\_t VertexesNumber() = 0;
9. virtual void Print(std::ostream& os) = 0;
10. **virtual double Area() = 0;**
11. virtual ~Figure() {};
12. };
14. #endif // FIGURE\_H

**point.h**

1. #ifndef POINT\_H
2. #define POINT\_H
4. #include <iostream>
6. class Point {
7. public:
8. Point();
9. Point(std::istream &is);
10. **Point(double x, double y);**
12. double dist(Point& other);
14. friend std::istream& operator>>(std::istream& is, Point& p);
15. **friend std::ostream& operator<<(std::ostream& os, Point& p);**
17. private:
18. double x\_;
19. double y\_;
20. **};**
22. #endif // POINT\_H

**point.cpp**

1. #include "point.h"
3. #include <cmath>
5. **Point::Point() : x\_(0.0), y\_(0.0) {}**
7. Point::Point(double x, double y) : x\_(x), y\_(y) {}
9. Point::Point(std::istream &is) {
10. **is >> x\_ >> y\_;**
11. }
13. double Point::dist(Point& other) {
14. double dx = (other.x\_ - x\_);
15. **double dy = (other.y\_ - y\_);**
16. return std::sqrt(dx\*dx + dy\*dy);
17. }
19. std::istream& operator>>(std::istream& is, Point& p) {
20. **is >> p.x\_ >> p.y\_;**
21. return is;
22. }
24. std::ostream& operator<<(std::ostream& os, Point& p) {
25. **os << "(" << p.x\_ << ", " << p.y\_ << ")";**
26. return os;
27. }

**octagon.h**

1. #ifndef OCTAGON\_H
2. #define OCTAGON\_H
4. #include <iostream>
6. #include "figure.h"
8. class Octagon : public Figure {
9. public:
10. **Octagon();**
11. Octagon(Point t\_1, Point t\_2, Point t\_3, Point t\_4,
12. Point t\_5, Point t\_6, Point t\_7, Point t\_8);
13. Octagon(std::istream &is);
14. Octagon(const Octagon& other);
16. size\_t VertexesNumber();
17. double Area();
18. void Print(std::ostream& os);
20. **virtual ~Octagon();**
22. friend std::istream& operator>>(std::istream& is, Octagon& o);
23. friend std::ostream& operator<<(std::ostream& os, Octagon& o);
25. **private:**
26. Point t1;
27. Point t2;
28. Point t3;
29. Point t4;
30. **Point t5;**
31. Point t6;
32. Point t7;
33. Point t8;
34. };
36. #endif // OCTAGON\_H

**octagon.cpp**

1. #include "octagon.h"
3. #include <iostream>
4. #include <cmath>
6. Octagon::Octagon()
7. : t1(0.0, 0.0), t2(0.0, 0.0), t3(0.0, 0.0), t4(0.0, 0.0),
8. t5(0.0, 0.0), t6(0.0, 0.0), t7(0.0, 0.0), t8(0.0, 0.0) {}
10. **Octagon::Octagon(Point t\_1, Point t\_2, Point t\_3, Point t\_4,**
11. Point t\_5, Point t\_6, Point t\_7, Point t\_8)
12. : t1(t\_1), t2(t\_2), t3(t\_3), t4(t\_4),
13. t5(t\_5), t6(t\_6), t7(t\_7), t8(t\_8) {}
15. **Octagon::Octagon(std::istream &is) {**
16. is >> t1 >> t2 >> t3 >> t4 >> t5 >> t6 >> t7 >> t8;
17. }
19. Octagon::Octagon(const Octagon& other)
20. **: Octagon(other.t1, other.t2, other.t3, other.t4,**
21. other.t5, other.t6, other.t7, other.t8) {}
23. std::istream& operator>>(std::istream& is, Octagon& o) {
24. is >> o.t1 >> o.t2 >> o.t3 >> o.t4 >> o.t5 >> o.t6 >> o.t7 >> o.t8;
25. **return is;**
26. }
28. std::ostream& operator<<(std::ostream& os, Octagon& o) {
29. os << "Octagon: " << o.t1 << " " << o.t2 << " " << o.t3 << " " << o.t4
30. **<< " " << o.t5 << " " << o.t6 << " " << o.t7 << " " << o.t8;**
31. return os;
32. }
34. size\_t Octagon::VertexesNumber()
35. **{**
36. return (size\_t)8;
37. }
39. double Heron(Point A, Point B, Point C) {
40. **double AB = A.dist(B);**
41. double BC = B.dist(C);
42. double AC = A.dist(C);
43. double p = (AB + BC + AC) / 2;
44. return sqrt(p \* (p - AB) \* (p - BC) \* (p - AC));
45. **}**
47. double Octagon::Area() {
48. double area1 = Heron(t1, t2, t3);
49. double area2 = Heron(t1, t4, t3);
50. **double area3 = Heron(t1, t4, t5);**
51. double area4 = Heron(t1, t5, t6);
52. double area5 = Heron(t1, t6, t7);
53. double area6 = Heron(t1, t7, t8);
54. return area1 + area2 + area3 + area4 + area5 + area6;
55. **}**
57. void Octagon::Print(std::ostream& os)
58. {
59. std::cout << "Octagon: " << t1 << " " << t2 << " " << t3 << " " << t4
60. **<< " " << t5 << " " << t6 << " " << t7 << " " << t8 << "\n";**
61. }
63. Octagon::~Octagon() {}

**item.h**

#ifndef ITEM\_H

#define ITEM\_H

#include "octagon.h"

class Item

{

public:

Item(const Octagon &s);

Item(const Item &other);

Item \*Left();

Item \*Right();

void ToLeft(Item \*node);

void ToRight(Item \*node);

Octagon& GetOctagon();

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

virtual ~Item();

private:

Octagon octagon;

Item \*prev;

Item \*next;

};

#endif // ITEM\_H

**item.cpp**

#include "item.h"

Item::Item(const Octagon &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;

}

Item \*Item::Left()

{

return this->prev;

}

Item \*Item::Right()

{

return this->next;

}

void Item::ToLeft(Item \*node)

{

this->prev = node;

}

void Item::ToRight(Item \*node)

{

this->next = node;

}

Octagon& 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**

#ifndef TLINKEDLIST\_H

#define TLINKEDLIST\_H

#include "item.h"

class TLinkedList

{

public:

TLinkedList();

TLinkedList(const TLinkedList &other);

const Octagon& First();

const Octagon& Last();

const Octagon& GetItem(size\_t idx);

size\_t Length();

bool Empty();

void InsertFirst(const Octagon& octagon);

void InsertLast(const Octagon& octagon);

void Insert(const Octagon& octagon, 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 \*beginning;

Item \*end;

};

#endif // TLINKEDLIST\_H

**tlinkedlist.cpp**

#include "tlinkedlist.h"

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

TLinkedList::TLinkedList(const TLinkedList &other)

{

beginning = other.beginning;

end = other.end;

}

const Octagon& TLinkedList::First()

{

if (beginning == nullptr) {

std::cout << "The list is empty" << std::endl;

exit(1);

}

return beginning->GetOctagon();

}

const Octagon& TLinkedList::Last()

{

if (end == nullptr) {

std::cout << "The list is empty" << std::endl;

exit(1);

}

return end->GetOctagon();

}

const Octagon& 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();

}

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

++size;

}

return size;

}

void TLinkedList::InsertFirst(const Octagon& octagon)

{

Item \*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(const Octagon& octagon)

{

Item \*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(const Octagon& 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;

}

Item \*node = new Item(octagon);

Item \*now = beginning;

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

now = now->Right();

}

Item \*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) {

delete beginning;

beginning = (end = nullptr);

return;

}

Item \*node = beginning;

beginning = beginning->Right();

beginning->ToLeft(nullptr);

delete node;

}

void TLinkedList::RemoveLast()

{

if (end == nullptr) {

std::cout << "The list is empty" << std::endl;

return;

}

if (end == beginning) {

delete beginning;

beginning = (end = nullptr);

return;

}

Item \*node = end;

end = end->Left();

end->ToRight(nullptr);

delete node;

}

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;

}

Item \*node = beginning;

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

node = node->Right();

}

Item \*node\_left = node->Left();

Item \*node\_right = node->Right();

node\_left->ToRight(node\_right);

node\_right->ToLeft(node\_left);

delete node;

}

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

{

if (list.beginning == nullptr) {

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

return os;

}

for (Item \*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);

Octagon o1(x1, x2, x3, x4, x5, x6, x7, x8);

Octagon o2(y1, y2, y3, y4, y5, y6, y7, y8);

Octagon o3(z1, z2, z3, z4, z5, z6, z7, z8);

//Octagon o1, o2, o3;

//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;

}