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

ЛАБОРАТОРНАЯ РАБОТА №5 по курсу

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

Студент *Зинин Владислав Владимирович, группа М80-208Б-20*
Преподаватель *Дорохов Евгений Павлович*

Цель работы

Целью лабораторной работы является:

Закрепление навыков работы с классами.

Знакомство с умными указателями.

Задание

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

Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.

Требования к классу контейнера аналогичны требованиям из лабораторной работы 2.

Класс-контейнер должен соджержать объекты используя std:shared_ptr<…>.

Классы должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Нельзя использовать:

Стандартные контейнеры std.

Шаблоны (template).

Объекты «по-значению»

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 10 файлах:

1. main.cpp - основная программа, взаимодействие с пользователем посредством команд из меню
2. include/figure.h - описание абстрактного класса фигур
3. include/point.h - описание класса точки
4. include/TVector.cpp - реализация функций контейнера первого уровня (в моем случае вектора)
5. include/TVector.h – реализация класса контейнера первого уровня (в моем случае вектора)
6. include/rhombus.h - описание класса ромба, наследующегося от figures
7. include/point.cpp - реализация класса точки
8. include/TVectorItem.cpp – реализация функций вспомогательного класса для контейнера
9. include/TVectorItem.h – описание вспомогательного класса для контейнера
10. include/rhombus.cpp: реализация класса ромба, наследующегося от figure

Дневник отладки

Во время выполнения лабораторной работы неисправностей почти не возникало, все было отлажено сразу же.

Недочёты

Недочётов не было обнаружено.

Выводы

Лабораторная работа №5 позволила мне полностью осознать концепцию умных указателей в языке C++ и отточить навыки в работе с ними. Всё прошло успешно.

Исходный код

Figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
#include <iostream>
#include "point.h"

class Figure
```

```
{
public:
virtual ~Figure(){};
virtual double Area() = 0;
virtual size_t VertexesNumber() = 0;
};
#endif //FIGURE_H
```

Point.cpp

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

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_;
}

std::istream& operator>>(std::istream& is, Point& p) {
is >> p.x_ >> p.y_;
return is;
}

std::ostream& operator<<(std::ostream& os, Point& p) {
os << "(" << p.x_ << ", " << p.y_ << ")";
return os;
}

double get_x(Point &other)
{
return other.x_;
}

double get_y(Point &other)
{
return other.y_;
```

```
}

void Point::set_x(Point &other, double x)
{
other.x_ = x;
}

void Point::set_y(Point &other, double y)
{
other.y_ = y;
}
```

Point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point
{
public:
Point();
Point(double x, double y);
Point(std::istream &is);
double dist(Point &other);
friend double get_x(Point &other);
friend double get_y(Point &other);
void set_x(Point &other, double x);
void set_y(Point &other, double y);
friend std::istream& operator>>(std::istream& is, Point& p);
friend std::ostream& operator<<(std::ostream& os, Point& p);

private:
double x_, y_;
};

#endif //POINT_H
```

TVector.cpp

```
#include <iostream>
#include "TVector.h"
```

```
/*----init----*/
TVector::TVector()
{
size = 0;
std::cout << "TVector created" << std::endl;
}

/*----bool----*/

bool TVector::Empty()
{
return size == 0?1:0;
}

void TVector::InsertLast(std::shared_ptr<Rhombus> &&rhomb)
{
std::shared_ptr<TVectorItem> value (new TVectorItem(rhomb));
if(size == 0)
{
this->first = value;
this->first->next = nullptr;
this->first = value;
size++;
}
else
{
std::shared_ptr<TVectorItem> end = this->first;
while(end->next != nullptr)
{
end = end->next;
}
end->next = value;
value->next = nullptr;
size++;
}
}
```

```
void TVector::Resize(const size_t new_size)
{
if(size == new_size || new_size < 1)
{
return;
}
else if(new_size > size)
{
size_t iter = new_size - size;
for(int i = 0; i < iter; i++)
{
InsertLast(std::shared_ptr<Rhombus>(new Rhombus()));
}
}
else{
size_t iter = new_size;
std::shared_ptr<TVectorItem> end = this->first;
for(int i = 0; i < iter; i++)
{
end = end->next;
}
end->next = nullptr;
}
size = new_size;
}

void TVector::RemoveLast()
{
if(size == 0)
{
std::cout << "List is empty" << std::endl;
}
else
{
if(size == 1)
```

```
{
size--;
std::shared_ptr<TVectorItem> del = this->first;
}
else
{
std::shared_ptr<TVectorItem> del = this->first;
std::shared_ptr<TVectorItem> save;
while(del->next != nullptr)
{
save = del;
del = del->next;
}
size--;
save->next = nullptr;
}
}
}

void TVector::Remove(size_t idx)
{
if(idx < 1 || idx > size)
{
std::cout << "Invalid erase!" << std::endl;
}
else
{
std::shared_ptr<TVectorItem> del;
std::shared_ptr<TVectorItem> prev_del;
std::shared_ptr<TVectorItem> next_del = this->first;
size--;
if(idx == 1)
{
del = this->first;
next_del = next_del->next;
```

```
this->first = next_del;
}
else
{
for(int i = 1; i < idx; ++i)
{
prev_del = next_del;
next_del = next_del->next;
}
del = next_del;
next_del = next_del->next;
prev_del->next = next_del;
}
}
}
/*-----Rhombus-----*/
const Rhombus& TVector::Last()
{
std::shared_ptr<TVectorItem> node = this->first;
while(node->next != nullptr)
{
node = node->next;
}
return *node->rhomb;
}
/*----destructor---*/
TVector::~TVector()
{
std::cout << "TVector deleted" << std::endl;
}
/*----size_t---*/
size_t TVector::Length()
{
return size;
```

```
}
/*----operator---*/

Rhombus& TVector::operator[](const size_t idx)
{
std::shared_ptr<TVectorItem> idx_rhomb = this->first;
for(int i = 1; i < idx; i++)
{
idx_rhomb = idx_rhomb->next;
}
return *idx_rhomb->rhomb;
}

std::ostream& operator<<(std::ostream& os, TVector& obj)
{
if(obj.size == 0)
{
os << "TList is empty" << std::endl;
}
else
{
os << "Print rhombus" << std::endl;
std::shared_ptr<TVectorItem> print = obj.first;
os << '[';
for(int i = 0; i < obj.size - 1; i++)
{
os << print->rhomb->Area() << " " << "," << " ";
print = print->next;
}
os << print->rhomb->Area() << ']';
os << std::endl;
}
return os;
}
```

TVector.h

```
#ifndef TVECTOR_H
```

```
#define TVECTOR_H
#include <iostream>
#include "TVectorItem.h"
#include "rhombus.h"
#include <memory>

class TVector
{
public:
/*-----init-----*/
TVector();
/*-----void-----*/
void Remove(size_t idx);
void Resize(const size_t new_size);
void InsertLast(std::shared_ptr<Rhombus> &&rhomb);
void RemoveLast();
/*-----Rhombus-----*/
const Rhombus& Last();
/*-----bool-----*/
bool Empty();
/*-----size_t-----*/
size_t Length();
/*----operator-----*/
Rhombus& operator[] (const size_t idx);
friend std::ostream& operator<<(std::ostream& os, TVector& obj);
/*-----destructor-----*/
~TVector();
private:
size_t size;
std::shared_ptr<TVectorItem> first;
};

#endif//TVECTOR_H
```

TVectorItem.cpp

```
#include <iostream>
#include "TVectorItem.h"
```

```
TVectorItem::TVectorItem(std::shared_ptr<Rhombus>& rhomb)
{
this->rhomb = rhomb;
this->next = nullptr;
}

std::ostream& operator<<(std::ostream& os, TVectorItem& obj)
{
os << obj.rhomb << " ";
return os;
}

TVectorItem::~TVectorItem()
{
std::cout << "TVectorItem deleted" << std::endl;
}
```

TVectorItem.h

```
#ifndef TVECTORITEM_H
#define TVECTORITEM_H
#include <iostream>
#include "rhombus.h"
#include <memory>

class TVectorItem
{
public:
TVectorItem(std::shared_ptr<Rhombus>& rhomb);
friend std::ostream& operator<<(std::ostream& os, TVectorItem& obj);
~TVectorItem();
std::shared_ptr<Rhombus> rhomb;
std::shared_ptr<TVectorItem> next;
};

#endif //TVECTORITEM_H
```

Main.cpp

```
#include <iostream>
#include "TVector.h"
```

```
int main()
{
TVector list;
/*-----Test push_front---*/
list.InsertLast(std::shared_ptr<Rhombus>(new Rhombus(Point(1,2), Point(3,4), Point(5,6), Point(7,8))));
list.InsertLast(std::shared_ptr<Rhombus>(new Rhombus(Point(1,3), Point(3,4), Point(5,6), Point(7,8))));
list.InsertLast(std::shared_ptr<Rhombus>(new Rhombus(Point(1,4), Point(3,4), Point(5,6), Point(7,8))));
list.InsertLast(std::shared_ptr<Rhombus>(new Rhombus(Point(1,5), Point(3,4), Point(5,6), Point(7,8))));
list.InsertLast(std::shared_ptr<Rhombus>(new Rhombus(Point(1,6), Point(3,4), Point(5,6), Point(7,8))));
list.InsertLast(std::shared_ptr<Rhombus>(new Rhombus(Point(1,7), Point(3,4), Point(5,6), Point(7,8))));
std::cout << list << std::endl;
/*-----Test pop_front---*/
list.RemoveLast();
std::cout << list << std::endl;

list.RemoveLast();
std::cout << list << std::endl;
/*-----Test push_back---*/
// list.push_front(std::shared_ptr<Rhombus>(new Rhombus(Point(2,3), Point(2,3), Point(2,3), Point(2,3))));
// std::cout << list << std::endl;
// /*-----Test pop_back---*/
// list.pop_front();
// std::cout << list << std::endl;
// /*-----Test clear---*/
// list.clear();
// std::cout << list << std::endl;

// list.push_front(std::shared_ptr<Rhombus>(new Rhombus(Point(2,3), Point(2,3), Point(2,3), Point(2,3))));
// std::cout << list << std::endl;
// /*-----Test insert---*/
// list.insert(std::shared_ptr<Rhombus>(new Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7))), 1);
// std::cout << list << std::endl;
// list.insert(std::shared_ptr<Rhombus>(new Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7))), 3);
// std::cout << list << std::endl;
// list.insert(std::shared_ptr<Rhombus>(new Rhombus(Point(0,1), Point(2,3), Point(4,5), Point(6,7))), 2);
```

```
// std::cout << list << std::endl;
/*-----Test erase---*/
list.Resize(2);
std::cout << list << std::endl;
std::cout << "----------" << std::endl;
std::cout << list.Length() << std::endl;
std::cout << list << std::endl;
std::cout << list[2] << std::endl;
list.Resize(4);
std::cout << list << std::endl;
list.Resize(4);
std::cout << list << std::endl;
return 0;
}
```

Rhombus.cpp

```
#include <iostream>
#include "rhombus.h"
#include <math.h>

Rhombus::Rhombus()
{
a.set_x(a, 1);
a.set_y(a, 1);
b.set_x(b, 2);
b.set_y(b, 2);
c.set_x(c, 0);
c.set_y(c, 3);
d.set_x(d, -1);
d.set_y(d, -1);
}

Rhombus::Rhombus(std::istream &is)
{
is >> a;
is >> b;
is >> c;
```

```
is >> d;
}

Rhombus::Rhombus(Point pa, Point pb, Point pc, Point pd): a(pa), b(pb), c(pc), d(pd)
{
std::cout << "Rhombus created" << std::endl;
}

// void Rhombus::Print(std::ostream &os)
// {
// os << "Rhombus" << std::endl;
// os << a << ',' << b << ',' << c << ',' << d << std::endl;
// }

double Rhombus::Area()
{
return 0.5 * fabs(get_x(a)*get_y(b) + get_x(b)*get_y(c) + get_x(c)*get_y(d) + get_x(d)*get_y(a) - get_x(b)*get_y(a)
- get_x(c)*get_y(b) - get_x(d)*get_y(c) - get_x(a)*get_y(d));
}

Rhombus::~Rhombus()
{
std::cout << "Rhombus deleted" << std::endl;
}

size_t Rhombus::VertexesNumber()
{
return 4;
}

std::ostream& operator<<(std::ostream& os, Rhombus& p) {
os << p.a << p.b << p.c << p.d;
return os;
}

Rhombus& Rhombus::operator=(const Rhombus& other) {
if (this == &other)
return *this;

this->a = other.a;
```

```
this->b = other.b;
this->c = other.c;
this->d = other.d;

std::cout << "Rhombus copied" << std::endl;

return *this;
}

// Point get_a(const Rhombus& other)
// {
// return other.a;
// }

// Point get_b(Rhombus& other)
// {
// return other.b;
// }

// Point get_c(Rhombus& other)
// {
// return other.c;
// }

// Point get_d(Rhombus& other)
// {
// return other.d;
// }
```

Rhombus.h

```
#ifndef RHOMBUX_H
#define RHOMBUX_H
#include <iostream>
#include "point.h"
#include "figure.h"

class Rhombus : public Figure
{
public:
Rhombus();
```

```
Rhombus(std::istream &is);
Rhombus(Point a, Point b, Point c, Point d);
double Area();
// void Print(std::ostream &os);
size_t VertexesNumber();
Rhombus& operator=(const Rhombus& other);
// friend Point get_a(Rhombus& other);
// friend Point get_b(Rhombus& other);
// friend Point get_c(Rhombus& other);
// friend Point get_d(Rhombus& other);
friend std::ostream& operator<<(std::ostream& os, Rhombus& p);
virtual ~Rhombus();
private:
Point a, b, c, d;
};
#endif //RHOMBUX_H
```